UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMATICA

APLICAÇÕES NA WEB PARA VALIDAÇÃO E CLASSIFICAÇÃO DE NOTÍCIAS PELA CROWD

Mestrado em Engenharia Informática
Especialização em Engenharia de Software

Tiago Miguel Marques Aparício

Trabalho de Projeto orientado por:
Professor Doutor Tiago João Vieira Guerreiro e
Professor Doutor Francisco José Moreira Couto

2015

Agradecimentos

Em primeiro lugar queria agradecer aos meus país pelo carinho, apoio e paciência que tiveram ao longo destes anos e que sem eles não seria possível chegar até aqui.

Agradeço ao meu professor e orientador, Tiago Guerreiro por toda a disponibilidade, dedicação e apoio fundamental para a realização desta dissertação.

Quero agradecer também aos meus colegas de faculdade, em especial à Ana Almeida, Carlos Barata, Mónica Abreu, Luís Rochinha, João Nascimento, João Martins, Rita Henriques, José Carilho, Fábio Santos, Gonçalo Semedo, pelo seu apoio e ajuda nas alturas em que mais necessitava, pelos momentos de diversão e palhaçada mas, principalmente pela sua amizade ao longo destes anos de faculdade.

Uma palavra de agradecimento para a Mariana Costa pelo apoio e ajuda que me deu na fase final desta dissertação e para a Alexandra Silva pelo apoio e jantar que me vai pagar por ter entregue a tese em Novembro.

À minha família e amigos

Resumo

Nos dias de hoje, existe uma quantidade enorme de notícias de transferências futebolísticas que são publicadas diariamente pelos meios de comunicação social. Ao lerem estes rumores, os leitores não conseguem dizer com objetividade se estes são fundamentados ou apenas boatos. Como forma de dar reposta a este problema, esta tese visa desenvolver mecanismos para ajudar na validação destas notícias de transferências futebolísticas. Sendo a *crowd* uma grande fonte de conhecimento e ao mesmo tempo fácil de aceder, torna-se um dos mecanismos importantes para a validação destas notícias. Assim, esta tese de mestrado tem como grande objetivo o desenvolvimento de mecanismos que permitam tirar partido do grande conhecimento que se encontra na *crowd* utilizando esse conhecimento para validar notícias sobre transferências futebolísticas.

Com esse objetivo em mente fez-se então o levantamento de requisitos que a aplicação tinha de cumprir e detalhou-se os casos de uso descrevendo esses requisitos que a aplicação deveria ter.

Para implementar as funcionalidades identificadas no levantamento de requisitos, criou-se um gráfico usando o *D3.js* para representar a evolução das notícias ao longo do tempo, uma página que permite os utilizadores validarem as notícias como concretizadas ou não concretizadas. Tirando partido das notícias recolhidas implementou-se filtros de pesquisa com o intuito de permitir ao utilizadores filtrar notícias por equipas e jogadores, através de *clicks* nos símbolos e fotos respetivamente. Implementou-se também estatísticas que permitem ao utilizador ter uma ideia sobre que equipas e jogadores se escrevem mais notícias. Por fim efetuou-se a integração com as redes sociais com recurso à ferramenta *Semantria* que permite analisar os comentários feitos pelos utilizadores no *Facebook*, classificando-os como positivos, negativos ou neutros. Utilizando essa classificação foi também criada uma página onde são listadas as notícias publicadas nas redes sociais com a respetiva análise de sentimento.

Após a implementação da aplicação, efetuaram-se testes de usabilidade com quinze utilizadores. Os resultados destes testes foram positivo, tendo em média, sido concluídas com sucesso 92% das tarefas propostas. No entanto, verificou-se que os utilizadores sentiram dificuldades em algumas tarefas, nomeadamente a tarefa 14, onde apenas 47% dos utilizadores concluíram a mesma.

Palavras-chave: Aplicações Web, Redes sociais, *Crowd*, Análise de sentimentos, Usabilidade

Abstract

Today, there is a huge amount of news about football transfers that are published daily by the media. Reading these rumors, one cannot objectively say if the rumor is true or not. In this line of thought, this thesis aims to develop mechanisms to aid in the validation of this news regarding football transfers. Since the crowd is both a great source of knowledge and also easy to access, it is considered one of the most important mechanisms used for the validation of these reports. Thus, the goal of this master's thesis is the development of mechanisms to validate news on football transfers taking advantage of the great knowledge that can be found among the crowd. To achieve this objetive a system will be developed that allows users to vote on news provided by the above mentioned system, thus helping in their validation.

With this goal in mind, the requirements that the application had to comply with were identified and the use cases describing them were detailed.

To implement the features identified in the requirements, it was created a graph using the D3.js to represent the evolution of news over time, a page that lets users validate the news as realized or unrealized. Taking advantage of the collected news it was implemented search filters in order to allow the users to filter news by teams and players, through clicks on the symbols and photos respectively. Statistics were also implemented thus allowing the user to get an idea of which teams and players are written more news. Finally, the integration with social networks was performed, taking advantage of the Semantria framework which analyzes the comments made by users on Facebook, classifying them as positive, negative or neutral. This classification was also used in a new page where the news published on social networks with the respective sentiment analysis are shown.

After the implementation of the application, it was performed usability tests with fifteen users. The results of these tests were positive, on average 92% of the proposed tasks were successfully completed. However, users had difficulties in some tasks, in particular the task 14 was only successfully concluded by 47 % of users.

Keywords: Web aplications, Social Networks, crowd, Sentiment Analysis, Usability

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A *Web*[10] nem sempre foi como a conhecemos nos dias de hoje. No inicio da sua criação, a *Web* era um local onde apenas um conjunto reduzido de pessoas partilhava conteúdos e a restante população consultava estes conteúdos. As paginas *Web*, eram portanto páginas de conteúdo estático, onde a única interação possível por parte do utilizador era a sua consulta. Este estado estático da *Web* foi denominado *Web* 1.0.

A evolução que se registou ao longo dos anos permitiu que qualquer utilizador da *Web* começasse a criar conteúdo tirando partido das ajudas tecnológicas existentes que facilitam essa criação e desenvolvimento, a esta "nova" *Web* chamou-se *Web* 2.0 à qual se junta um conjunto de inovações que ocorreram na *Web* ao longo dos anos. Embora ainda seja difícil fazer uma definição precisa da mesma, é possível fazer uma separação dos *sites* existentes como pertencentes à *Web* 2.0, como é o caso do *Facebook*, *Youtube* e os que pertencem à Web 1.0, como *ALIWEB*[1]. Esta separação é suportada pelas diferenças visíveis ao nível da tecnologia, estrutura e da componente sociológica dos referidos *sites*. Para ajudar a classificar um *site* como pertencente à *Web* 2.0 definiu-se o seguinte conjunto de características:

- Os utilizadores são o foco de sistema, através da criação de perfis proeminentes;
- A habilidade de formar ligações entre os utilizadores;
- A habilidade de postar conteúdo de várias formas[2] e controlar a privacidade e partilha;
- Fornecer características mais técnicas como uma API pública que permita melhoramentos por parte de terceiros e *mash-ups*.

As inovações tecnológicas da *Web* 2.0 levaram ao aparecimento das redes sociais e *sites* com uma forte componente social onde é possível criar nichos, ou seja, grupos de

[1] http://www.aliweb.com/

[2] fotos, videos, *blogs*, comentar e classificar conteúdo de outros utilizadores e identificar conteúdo próprio e de outros

amigos entre os quais se pode trocar conteúdo de qualquer tipo, seja este texto, áudio ou vídeo, efetuar *tags*, comentários e *links* para páginas internas ou externas ao *site*.

Outra das preocupações que foram surgindo ao longo dos anos, foi a falta de interatividade por parte das aplicações *Web* que, apesar de serem populares, eram pouco responsivas e interativas para as necessidades dos utilizadores. A interação clássica nas aplicações *Web* é baseada num interface com múltiplas páginas, em que cada novo pedido obriga a um novo carregamento do mesmo. Recentemente, os avanços tecnológicos ao nível da *Web* levaram a criação de uma nova técnica para criar aplicações *Web* interativas, chamada *Asynchronous JavaScript And XML* (*AJAX*). O aparecimento desta tecnologia permitiu a criação de *Single-Page Applications* (*SPA*)[16], que são aplicações compostas por um interface com uma única página formada por componentes individuais que são carregados dinamicamente a cada iteração do utilizador de modo a que a página inteira não tenha de se atualizar de cada vez que o utilizador interage com a mesma, ou seja, as *SPA* incorporam o conteúdo de várias páginas numa única, sendo apenas visível um dos componentes de cada vez. Sempre que o utilizador interage com a página, é carregado um novo componente sobre o anterior entretanto removido. Isto permite aumentar a interatividade, responsividade e satisfação dos utilizadores para com a aplicação.

1.1 Motivação

O aparecimento da *internet* veio revolucionar a maneira como é propagada a informação. Atualmente, a maior parte dos dispositivos tem a possibilidade de se ligar à *internet*,consequentemente, esta informação chega de forma mais fácil a um maior número de pessoas. Esta difusão é também feita em grande parte nas redes sociais nas quais os utilizadores partilham, por exemplo, notícias com o seu grupo de amigos e/ou seguidores.

Devido a esta facilidade de partilha de informação através da *internet* existe uma grande corrente de informação, seja através da comunicação social, de *blogs* pessoais, redes sociais, entre outros. No entanto, é de notar que estes fatores levam a um aumento da quantidade de informação disponível o que dificulta o processamento da mesma. O problema toma outras proporções quando chegamos à comunicação social desportiva, uma vez que é uma área na qual existe muita especulação devido a vários fatores. Por exemplo considerando hipoteticamente que, um empresário a tentar arranjar para um jogador seu um melhor contrato ou uma transferência para um clube com maior reputação , isto pode levar a que surjam notícias de que um certo clube está interessado nesse jogador, sem que isto seja necessariamente verdade.

Para ajudar a responder a este problema, procurar-se-á tirar proveito do conceito de *crowd* e de todo o conhecimento que pode ser daí retirado. Neste caso, a *crowd* é representada pelos adeptos de futebol que possuem conhecimento e opiniões que podem ser

tidas em conta para validar e classificar a veracidade das notícias publicadas pelas fontes jornalísticas.

1.2 Objetivos

Esta tese tem como objetivo o desenvolvimento de uma aplicação *Web* que permita uma validação de notícias sobre transferências futebolísticas por parte da *crowd*. Este objetivo pode ser dividido da seguinte forma:

1. Recolha de notícias sobre transferências futebolísticas para posterior validação.
2. Realização de um sistema que permita a validação de dados previamente recolhidos.
3. Concretização de um sistema para visualização de variadas estatísticas sobre os dados, como por exemplo, que jogadores/clubes geram mais rumores, etc.
4. Efetuar um sistema que mostre as notícias desportivas ao longo do tempo e que permita o uso de vários filtros por notícia.
5. Realização de um sistema que será responsável pela análise de sentimentos nas redes sociais, em relação às notícias que vão sendo colocadas pelas fontes jornalísticas desportivas, mostrando para cada notícia qual a reação provocada nos leitores.

Esta dissertação teve como resultado a aplicação *Web* chamada *Rankores*[3].

1.3 Estrutura do documento

Este documento está organizado da seguinte forma:

- Capítulo 2 - Trabalho relacionado: Apresenta o contexto onde esta tese se insere e os trabalhos relacionados realizados nesta área.
- Capítulo 3 - Análise e Desenho: Levantamento de requisitos para o projeto
- Capítulo 4 - Implementação: Explicação sobre o desenvolvimento do projeto
- Capítulo 5 - Resultados: Mostrar os resultados obtidos
- Capítulo 6 - Conclusão: Descrição do que foi concluído através da realização desta tese

[3] `http://www.lasige.di.fc.ul.pt/webtools/Rankores/`

Capítulo 2

Trabalho relacionado

Neste capítulo irá ser apresentada uma pequena descrição de projetos dentro do mesmo âmbito desta dissertação. Irá ser referido que tecnologias foram usadas por estes projetos e de que modo é que esse *know-how* pode ser aplicado nesta dissertação.

2.1 Projetos relacionados

2.1.1 *TwitterEcho*

As redes sociais que apareceram ao longo dos últimos anos, modificaram a maneira como comunicamos e consequentemente tornaram-se objeto de análises de dados, como extração de informação, recolha de opiniões, etc. Os dados por detrás destes serviços são vastos e continuam a aumentar a um ritmo elevado, o que as torna bastante atrativas a este tipo de análises. O *Twitter* é um dos maiores exemplos deste tipo de serviço, sendo uma plataforma de *micro-bloging*, tem o potencial para ser uma grande fonte de informação, como opiniões, ideias, factos e sentimentos.

O *TwitterEcho*[5][1] tem como objetivo desenvolver uma camada de análise de dados. Inicialmente, focou-se nos módulos de análise das redes sociais, ou seja, filtros de pré-processamento de texto, classificação de linguagem, recolha de opiniões, análise estatística dos tópicos mais falados e deteção de influenciadores. Este projeto foca-se igualmente no desenvolvimento de visualizações de dados para caracterizar padrões de atividade e comportamentos dos utilizadores na comunidade *Twitter*.

2.1.2 *Social Bus*

As redes sociais oferecem uma API[2] que permite aceder ao que é partilhado publicamente pelos seus utilizadores, contudo o acesso a esta informação está muitas vezes limitado por estas. Grande parte destas limitações prendem-se com o facto de apenas ser permitido um

[1] `http://labs.sapo.pt/2012/12/twitterecho-3-data-analytics-feup/`
[2] ***Api* do *Facebook*:** `https://developers.facebook.com`
***Api* do *Twitter*:** `https://dev.twitter.com`

certo número de pedidos às *APIs* durante um certo período de tempo. Para vários projetos de investigação é essencial o uso desta informação. Muitas vezes é necessário o acesso a dados produzidos anteriormente para fazer estudos sobre esta informação. Assim sendo, o uso direto das *APIs* pode não ser suficiente.

O "*Social Bus* [3]"é a continuação do projeto "*TwitterEcho*", sendo esta *framework* uma atualização do deste em que a principal diferença entre este dois projetos é a adição da funcionalidade de recolha de informação no *Facebook*. Ambos os projetos funcionam através de *crawlers* que estão constantemente a recolher o que é partilhado nas duas redes sociais. Estes *crawlers* são denominados como consumidores, visto que consumem os dados das redes sociais, guardando-os num repositório. Existem, portanto, dois consumidores:

1. O consumidor do *Twitter* é usado para monitorizar e recolher as mensagens trocadas nesta rede social, que são chamadas *tweets*. Os utilizadores podem definir quais os termos de pesquisa que devolvem *tweets* que os refiram. É também possível recolher todos os *tweets* feitos por determinados utilizadores ao longo do tempo.

2. O consumidor do *Facebook* é bastante parecido com o do *Twitter* com a diferença que apenas recolhe as mensagens (*posts*) públicas, ao contrário do *Twitter* que consegue recolher todos, porque todos os *tweets* são públicos. A figura 2.1 representa a arquitetura do *Social Bus*.

2.1.3 Mundo em Pessoa

"O Mundo em Pessoa" [4] foi um projeto de recolha automática de citações de Fernando Pessoa (ortónimos e heterónimos) a partir das redes sociais mais utilizadas (*Facebook* e *Twitter*). O projeto foi lançado em 2013, na altura da comemoração dos 125 anos do nascimento do poeta. Sendo este o poeta português mais conhecido dentro e fora de Portugal, é também provavelmente o mais citado. Com este projeto, foi possível identificar quais os versos e frases de Fernando Pessoa que mais inspiram os seus leitores de todo o mundo. Outro objetivo foi, conduzir todos aqueles que usam as palavras de Pessoa até ao seu texto original, ampliando o número de leitores e o conhecimento da sua obra. Sempre que é citado um texto de Fernando Pessoa no *Twitter*, em página ou figuras públicas do *Facebook*, "O Mundo em Pessoa" identifica e mostra essa mensagem num interface próprio. Para validar se um texto é uma citação da obra de Fernando Pessoa, este é comparado com arquivos da obra do poeta disponíveis *online*. Este projeto será importante, uma vez que se podem aproveitar as técnicas utilizadas para retirar informação das redes sociais.

[3] `http://reaction.fe.up.pt/socialbus/`
[4] `http://fernandopessoa.labs.sapo.pt/`

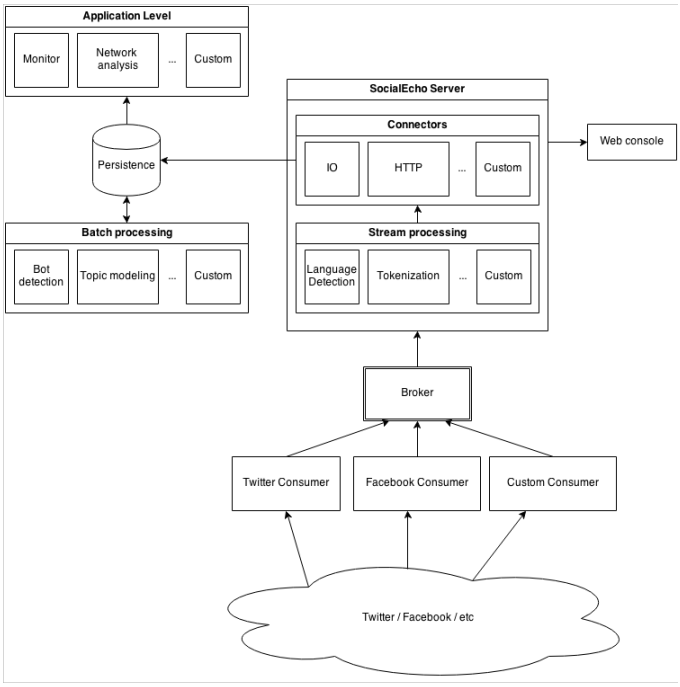

Figura 2.1: Arquitetura do Social Bus

2.1.4 Lusica

O "Lusica"[5] é um projeto de recolha automática de citações de músicas de artistas lusófonos a partir das redes sociais. O objetivo inicial é produzir um historial da popularidade dos estilos de música típicos da lusofonia (ex.: fado, samba, etc.) nas redes sociais. Desta forma, o projeto pretende promover a divulgação dos estilos, artistas e músicas pela comunidade. Utilizando o projeto *Social Bus* descrito na secção 2.1.2, recolheu-se um conjunto de *tweets* portugueses desde 2011 até à atualidade. Deste conjunto, é feita uma pesquisa pelo nome dos artistas lusófonos mais populares (segundo o site Last FM). A validação da citação é feita através de algoritmos de recolha de informação disponibilizados pela ferramenta *Apache Lucene*. O protótipo inicial baseia-se apenas na comparação dos *tweets* recolhidos com o título que foi obtido através do site *LastFM*[6]. Para mostrar as referências a cada estilo de música ao longo do tempo, é gerado um gráfico que nos mostra as referências a cada estilo de música que foi gerado com base na biblioteca *Data Driven Documents* (*D3.js* - capítulo 4.2.3). Todos os dados são obtidos pelos serviços *Web*, através dos quais se consegue saber os álbuns que foram citados num determinado tempo, de um estilo específico e visualizar a informação sobre os mesmos. Consegue-se ainda saber quais são os artistas, álbuns e músicas mais citadas ao longo dos vários anos.

[5] `http://www.lasige.di.fc.ul.pt/webtools/lusica/index.html`
[6] http://www.last.fm/pt/

2.1.5 Grande Área

O projeto "Grande Área" [7] é um projeto que visa a exploração interativa de um repositório histórico de informação desportiva. Focado no futebol, este projeto foi desenvolvido pelo SAPO Labs e apresentado publicamente na 7ª edição do *Codebits*. Organiza a informação de acordo com três pilares base: golos, cartões e minutos jogados. Os jogadores e equipas das principais ligas europeias podem ser ordenados de acordo com cada uma destas dimensões, destacando-se aqueles que obtiveram melhores resultados para a dimensão em questão. Adicionalmente, existe uma linha temporal, que permite analisar a evolução dos jogadores ou equipas ao longo dos últimos 7 anos. Páginas de perfil dos jogadores e equipas dão a possibilidade de analisar mais detalhadamente estatísticas dos jogos, transferências de jogadores, carreira profissional do jogador e vídeos com os melhores momentos dos jogos da Liga Portuguesa desde 2008. Este projeto será importante, uma vez que usa técnicas para representar dados de uma maneira agradável e de fácil navegação para o utilizador.

2.1.6 *Twitómetro*

O "*Twitómetro*" [8], é uma ferramenta que permite aferir o sentimento dos portugueses relativamente aos cinco líderes partidários com representação parlamentar durante as eleições parlamentares de 2011.

O *Twitómetro* era atualizado diariamente, após análise das mensagens que estão a ser recolhidas em permanência da rede social *Twitter*. A análise consiste na identificação de um dos líderes políticos alvo, seguida da deteção da polaridade do sentimento expresso em cada uma dessas mensagens (positiva ou negativa).

A partir desta análise, produziram-se, para cada dia, estatísticas que resumem as tendências relativas de cada candidato e entre os candidatos ao longo do tempo. Assim, no gráfico da figura 2.2, para cada dia e para cada candidato teremos uma circunferência colorida que resume duas dimensões da análise:

- A popularidade, medida em função do número relativo de menções que cada candidato tem no *Twitter*, é representada pela área da circunferência. Quanto maior for a circunferência, maior é a presença relativa de cada candidato nas mensagens da rede social *Twitter*;

- A tendência global do sentimento expresso nas mensagens *Twitter*, em função do número de mensagem positivas e negativas que foram detetadas para cada candidato. Note-se que, quanto mais mencionados forem os candidatos, mais expressivas serão as oscilações nas correspondentes posições verticais das suas circunferências.

[7] `http://grandearea.desporto.sapo.pt/TodosMundiais/Estatisticas/TopSelecoes/Eficiencia/`

[8] `http://legislativas.sapo.pt/2011/twitometro/`

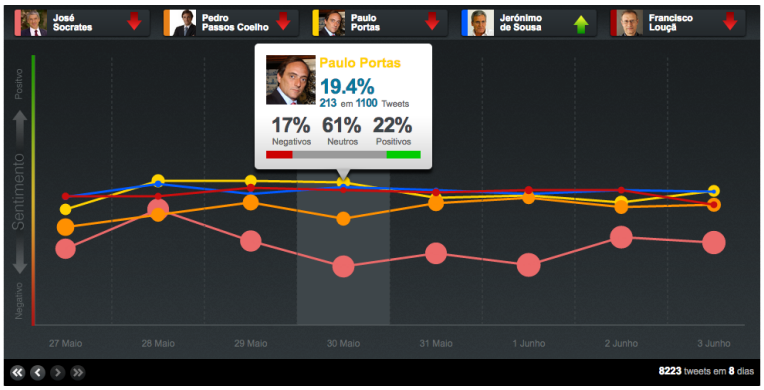

Figura 2.2: *Twitómetro*

Capítulo 3

Análise

Nesta capítulo serão descritos os requisitos necessários para a implementação deste projeto de dissertação.Para o efeito serão listados os requisitos funcionais e não funcionais.

3.1 Requisitos Funcionais

Os requisitos funcionais focam-se na funcionalidade e os serviços do sistema, ou seja, nas funções que o sistema deve fornecer ao utilizador e como o sistema se deve comportar para garantir essas funcionalidades.

Para alcançar cada objetivo referido na secção 1.2 foram definidos requisitos funcionais do sistema. Estes encontram-se enumerados de seguida:

- Visualizar notícias ao longo do tempo:
  - Visualizar todas as notícias numa *timeline*.
  - Filtrar as notícias na *timeline* por equipas.
  - Listar as notícias de uma data equipa ao longo de um período de tempo;
- Validar notícias antigas. Permitir aos utilizadores classificar uma notícia, isto é, dizer se a notícia se concretizou ou não, se é transferência ou não.
- Filtrar notícias por:
  - jogadores,
  - equipas,
  - notícias concretizadas,
  - e não concretizadas;
- Visualizar estatísticas sobre:
  - as equipas com mais notícias,

  - os jogadores com mais notícias,
  - e os jornais com mais notícias concretizadas

- Visualizar a reação das pessoas às notícias, ou seja, a aplicação deve conseguir mostrar a reação que as notícias partilhadas nas redes sociais provocaram nas pessoas.
- Visualizar os comentários feitos à notícia nas redes sociais.

3.2 Requisitos não Funcionais

Os requisitos não funcionais referem-se a critérios que podem ser usados para avaliar as operações de um sistema, em vez de comportamentos específicos. Os requisitos não-funcionais para este projeto, são os seguintes:

- **Usabilidade**: o interface da aplicação deverá ser intuitiva, fácil de usar, e de aprender a utilizar as funcionalidades que a aplicação suporta.
- **Fiabilidade**: a aplicação deve apresentar o mínimo de falhas possíveis na sua utilização.
- **Disponibilidade**: a aplicação deverá ter alta disponibilidade nas funcionalidades que apresenta.
- **Portabilidade**: a aplicação deverá estar disponível em qualquer dispositivo, tanto *desktop* como *mobile*.
- **Manutenção**: a aplicação deverá ser de fácil manutenção, permitindo uma fácil atualização. Deverá permitir uma fácil adaptação a novos requisitos.

3.3 Caso de Uso

O diagrama de caso de uso descreve a funcionalidade proposta para um novo sistema que será projetado, sendo esta uma excelente ferramenta para representar todas as interações possíveis, entre o utilizador e o sistema. Estes diagramas são usados para descrever a sequência de eventos que um ator que usa a aplicação deverá seguir para completar um processo.

Sistema

O sistema representa, neste caso, a aplicação *Web Rankores*.

Ator

O ator especifica o papel executado por um utilizador ou outro sistema que interage com o sistema. Este deve ser externo ao sistema, deve ter associações exclusivamente para casos de uso, componentes ou classes e é representado por um boneco (*stick man*).

No caso específico desta dissertação, o ator terá o papel de utilizador que irá interagir com o sistema.

Caso de uso

É uma especificação de um conjunto de ações executadas por um sistema, que contém um resultado observável. É representado por uma elipse, com o nome deste dentro ou abaixo. Num diagrama de casos de uso, os casos de uso estão associados ao ator que os executa.

Com base nos requisitos funcionais levantados na secção 3.1, foram definidos os seguintes caso de uso ilustrados na figura 3.1.

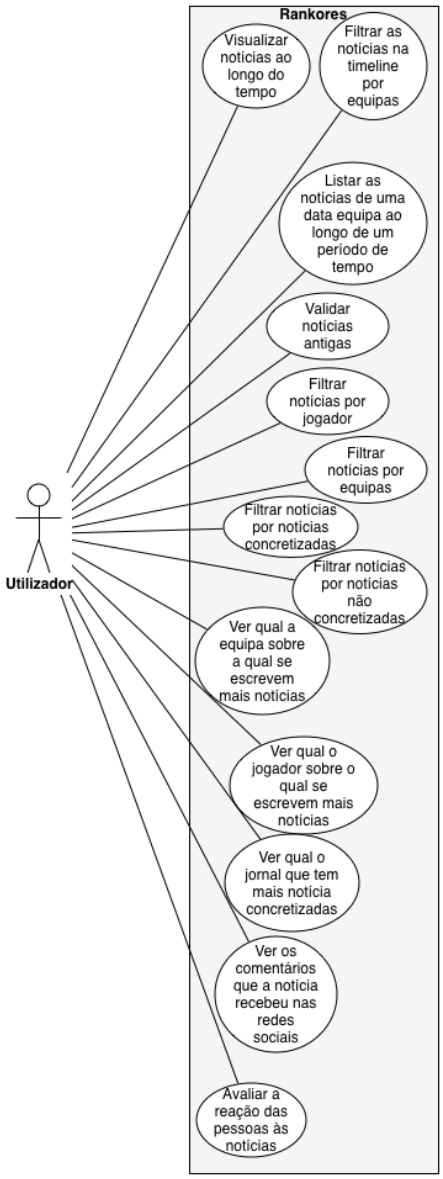

Figura 3.1: Diagrama de Casos de Uso da aplicação *Rankores*

Para cada caso de uso mencionado na figura 3.1 são descritos, nas subsecções seguin-

tes, os objetivos, atores, fluxo de ações, o resultado previsto e para alguns casos o estado de erro associado.

3.3.1 Visualizar notícias ao longo do tempo

- **Objetivos:** Visualizar notícias no espaço temporal da *timeline*.
- **Ator:** Utilizador.
- **Fluxo de ações:** Aceder à aplicação.
- **Resultado previsto:** O utilizador conseguir ver um gráfico que mostra a variação de notícias ao longo do tempo.

3.3.2 Filtrar as notícias na *timeline* por equipas

- **Objetivos:** Mostrar notícias das equipas selecionadas pelo utilizador no espaço temporal da *timeline*.
- **Ator:** Utilizador.
- **Fluxo de ações:**
  1. Aceder à aplicação,
  2. Selecionar as equipas quer pretende nas *drop-lists*.
  3. Confirmar a escolha.
- **Resultado previsto:** O utilizador conseguir ver um gráfico que mostra a variação de notícias ao longo do tempo para as equipas selecionadas.
- **Estado de erro:** Escolheu duas vezes a mesma equipa.

3.3.3 Listar as notícias de uma dada equipa ao longo de um período de tempo

- **Objetivos:** Listar as notícias de uma equipa,escolhida pelo utilizador, no período desejado.
- **Ator:** Utilizador.
- **Fluxo de ações:**
  1. Aceder à aplicação,
  2. Selecionar o espaço temporal desejado fazendo *scroll* no gráfico,

3. Clicar na linha da equipa desejada.

- **Resultado previsto:** O utilizador conseguir ver todas as notícias da equipa para o período desejado.

3.3.4 Validar notícias antigas

- **Objetivos:** Validar notícias antigas.
- **Ator:** Utilizador.
- **Fluxo de ações:**

1. Aceder à aplicação,
2. Selecionar a pagina "Votar",
3. Escolher que notícia quer validar e clicar no botão "Ler mais",
4. Selecionar uma das opções de resposta apresentada.

- **Resultado previsto:** A validação do utilizador é inserida na base de dado e o utilizador é reencaminhado para a lista de notícias para continuar a validação.

3.3.5 Filtrar notícias por jogador

- **Objetivos:** Mostrar todas as notícias do jogador selecionado.
- **Ator:** Utilizador.
- **Fluxo de ações:**

1. Aceder à aplicação,
2. Selecionar a página "de Jogadores",
3. Clicar no símbolo do clube a que o jogador pertence,
4. Selecionar a foto do jogador desejado.

- **Resultado previsto:** São listadas as notícias sobre o jogador selecionado.

3.3.6 Filtrar notícias por equipas

- **Objetivos:** Mostrar todas as notícias da equipa selecionada pelo utilizador.
- **Ator:** Utilizador.
- **Fluxo de ações:**

1. Aceder à aplicação,
2. Selecionar a pagina “Votar”,
3. Clicar no símbolo do clube sobre o qual pretende ver notícias.

- **Resultado previsto:** São listadas as notícias sobre a equipa selecionada.

3.3.7 Filtrar notícias por notícias concretizadas

- **Objetivos:** Mostrar todas as notícias concretizadas.
- **Ator:** Utilizador.
- **Fluxo de ações:**

1. Aceder à aplicação,
2. Selecionar a página “Concretizadas”.

- **Resultado previsto:** São listadas as notícias que foram votadas como concretizadas.

3.3.8 Filtrar notícias por notícias não concretizadas

- **Objetivos:** Mostrar todas as notícias não concretizadas.
- **Ator:** Utilizador.
- **Fluxo de ações:**

1. Aceder à aplicação,
2. Selecionar a página “Não concretizadas”.

- **Resultado previsto:** São listadas as notícias que foram votadas como não concretizadas.

3.3.9 Ver qual a equipa sobre a qual se escrevem mais notícias

- **Objetivos:** Mostrar um *top* de equipas com maior número de notícias.
- **Ator:** Utilizador.
- **Fluxo de ações:**

1. Aceder à aplicação,
2. Selecionar a página “Estatísticas”,
3. Clicar no botão equipas.

- **Resultado previsto:** São listadas as equipas com maior número de notícias.

3.3.10 Ver qual o jogador sobre o qual se escrevem mais notícias

- **Objetivos:** Mostrar um *top* de jogadores com maior número de notícias.
- **Ator:** Utilizador.
- **Fluxo de ações:**
  1. Aceder à aplicação,
  2. Selecionar a página "Estatísticas",
  3. Clicar no botão jogadores.
- **Resultado previsto:** São listadas os jogadores com maior número de notícias escritas.

3.3.11 Ver qual o jornal com mais notícias concretizadas

- **Objetivos:** Mostrar um *top* de jornais com mais notícias concretizadas.
- **Ator:** Utilizador.
- **Fluxo de ações:**
  1. Aceder à aplicação,
  2. selecionar a pagina "Estatísticas",
  3. clicar no botão jornais.
- **Resultado previsto:** São listadas os jornais com mais notícias concretizadas.

3.3.12 Avaliar a reação das pessoas às notícias

- **Objetivos:** Mostra as reações das pessoas sobre a notícia selecionada.
- **Ator:** Utilizador.
- **Fluxo de ações:**
  1. Aceder à aplicação,
  2. Selecionar a página "Redes Sociais",
  3. Escolher a notícia sobre a qual quer ver a reação
  4. Clicar "Ver mais".
- **Resultado previsto:** É mostrada a reação que a notícia causou.

- **Estado erro:** Se o utilizador seleciona uma notícia do *Twitter* que não tem qualquer comentário associado nesta rede social, a ferramenta não será capaz de apresentar qualquer resultado, uma vez que o cálculo da reação das pessoas depende da existência de comentários na notícia.

3.3.13 Ver os comentários que a notícia recebeu nas redes sociais

- **Objetivos:** Apresentar os comentários realizados à notícia nas redes sociais.
- **Ator:** Utilizador.
- **Fluxo de ações:**
  1. Aceder à aplicação,
  2. Selecionar a pagina "Redes Sociais",
  3. Escolher a notícia sobre a qual quer os comentários,
  4. Clicar "Ver mais",
  5. Clicar no botão "+"em frente ao comentários.
- **Resultado previsto:** São listados todos os comentários que essa notícia recebeu.
- **Estado erro:** Se o utilizador seleciona uma notícia do *Twitter* que não tem qualquer comentário associado nesta rede social, a ferramenta não será capaz de apresentar qualquer resultado.

3.4 Esboços da aplicação

Uma vez concluída a fase de levantamento de requisitos, é necessário idealizar como a aplicação irá responder a esses requisitos e que aspeto irá ter após a implementação dos mesmos. Com esse objetivo desenharam-se esboços da interface da aplicação.

A figura 3.2 mostra um primeiro esboço para implementar o caso de uso referido na secção 3.3.1. Este consiste em permitir ao utilizador visualizar a variação das notícias sobre as equipas ao longo do tempo. Pensou-se em ter uma barra com um intervalo de datas e que para cada data selecionada pelo utilizador listavam um conjunto de notícias para essa data.

Aquando do inicio do desenvolvimento verificou-se que faria mais sentido usar um gráfico com linhas, maneira a representar a evolução das notícias ao longo do tempo uma vez que permite uma visualização mais agradável aos utilizadores.

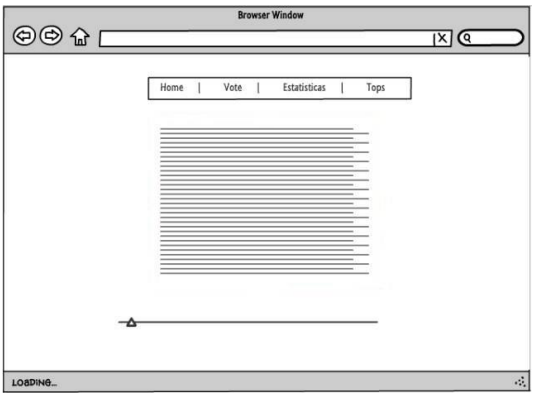

Figura 3.2: Esboço da *timeline* da aplicação *Rankores*

A figura 3.3 representa a ideia inicial para a realização da pagina de estatísticas. Primeiramente pensou-se em utilizar um gráfico de barras para representar o numero de notícias relativas a equipas e jogadores.No entanto,uma vez que existe um elevado número jogadores e equipas isso seria incomportável. Optou-se então, como alternativa em representar as estatísticas como um *top* ordenado do jogador/equipa mais citado para o menos citado.

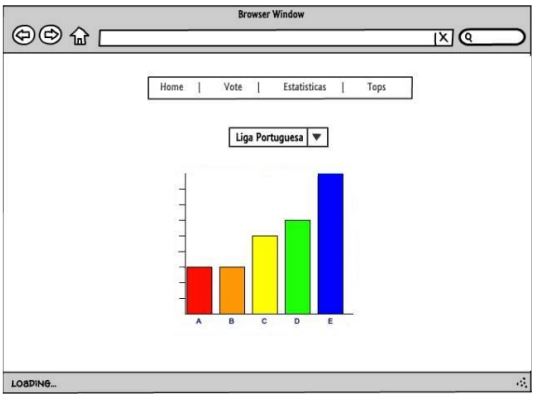

Figura 3.3: Esboço da página de estatísticas da aplicação *Rankores*

Na figura 3.4 encontra-se representado a ideia para aplicar filtros de pesquisa. Como se pode ver na figura essa ideia consiste numa barra de procura, na qual se podia pesquisar por qualquer jogador e equipa.

No entanto na fase de implementação optou-se por outro caminho, para implementar esta funcionalidade permitindo também dar resposta aos casos de usos identificados nas secções 3.3.5 e 3.3.6. Este novo caminho resultou num filtro de pesquisa através de *clicks* nas imagens das equipas e jogadores, de modo a que se assemelhasse ao que os utilizadores encontram nos *sites* de jornais desportivos em Portugal.

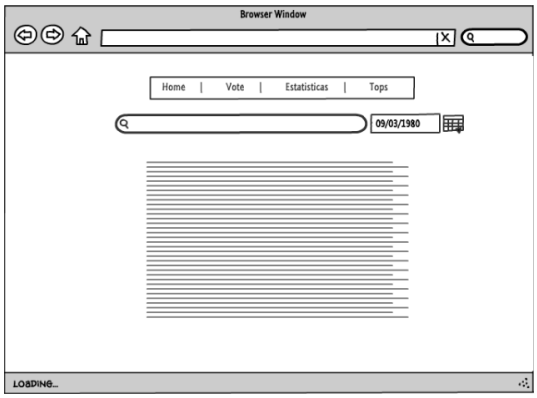

Figura 3.4: Esboço da página de estatísticas da aplicação *Rankores*

Capítulo 4

Implementação

Neste capítulo será apresentada uma descrição o do processo de implementação do projeto, começando com uma apresentação da tecnologia usada, seguindo-se uma explicação sobre a arquitetura da aplicação e da base de dados e para concluir como está organizada a estrutura da aplicação.

4.1 Conceitos

Nesta secção vão ser descritos conceitos úteis para a concretização da aplicação resultante desta dissertação.

4.1.1 *Wisdom of the crowd*

Segundo *Sheng Kung Michael Yi*[28], quando se resolve um problema usando o conhecimento de um grupo de pessoas, a solução que se obtém com o conjunto das deliberações individuais é geralmente tão boa ou até mesmo melhor do que se fosse considerada apenas a avaliação da melhor pessoa do grupo. Este fenómeno é descrito como *Wisdom of the crowd* e assenta na capacidade de filtrar ruído das opiniões individuais de forma a chegar mais perto da verdade. Existem dois métodos para combinar as análises individuais:

1. O primeiro método identifica os aspetos comuns das soluções individuais, sendo estes combinados para gerar uma solução válida.

2. O segundo método, em vez de fazer a decomposição da solução, identifica a solução individual que é mais parecida com outras soluções individuais. Contudo, este método não consegue formar uma solução que seja melhor que as soluções individuais.

A *Wikipedia*[18] é considerada o maior exemplo da utilização deste conceito, apesar de vários investigadores questionarem se uma enciclopédia que é produzida por vários utilizadores anónimos, resulta em informação de qualidade. Foi também questionado o facto

de a *Wikipedia* afimar ser precisa e neutra, apontando como falha o uso de contribuidores anónimos cujas intenções e perícia não são conhecidas. Estas questões, apesar de serem legítimas só têm em conta o fator humano, negligenciando a parte tecnológica associada. Para responder as estas preocupações, foram feitas várias análises em [18]:

1. Em primeiro lugar, analisou-se a questão de a *Wikipedia* ser um produto que resulta da escrita de alguns peritos ou de vários contribuidores comuns. Para responder a isto, foi analisada a sua estrutura hierárquica. Esta estrutura agrupa administradores, utilizadores registados, utilizadores anónimos e *bots* de maneiras distintas com base nas suas permissões.

2. De seguida, estudou-se a qualidade da informação, ou seja, se esta é precisa e neutra. De forma a demonstrar a qualidade da mesma, analisou-se os protocolos e tecnologias aplicadas para facilitar a edição consensual.

3. Em último lugar, foi demonstrada a dependência que os vários grupos de utilizadores têm dos agentes não-humanos (*bots*) que ajudam o processo de edição.

Como resultado de estudar estas questões, concluiu-se que o conteúdo da *Wikipedia* é mantido com base na colaboração entre utilizadores e *bots*, sendo estes últimos usados para detetar e anular atos de vandalismo e caso seja necessário, banir utilizadores. Os *bots* têm também um papel importante na criação e manutenção de novas entradas. Em 2009, a *Wikipedia* introduziu a *WikiTrust* que é uma extensão que classifica novas edições por cores tendo em conta a fiabilidade baseada na reputação do utilizador. Esta reputação é estabelecida de acordo com a longividade das outras contribuições. A *Wikipedia* em vez de se virar para peritos para classificar os seus artigos, usa uma combinação de regras, hierarquias e editores. Este conceito é útil neste projeto, na medida em que, usando a opinião das pessoas, ajudará a identificar se as notícias foram concretizadas ou não.

4.1.2 *Crawlers*

Segundo *Junghoo* [9], um *crawler* é um programa que recolhe páginas *web* de modo a criar um índice local. Um *crawler* começa com um conjunto de *URLs* chamados *seed URLs*, obtendo as páginas dos mesmos e recolhendo quaisquer *URLs* que a página contenha, adicionando-os numa fila de *URLs* para serem analisados. O processo é repetido para os *URLs* da fila. Normalmente, um *crawler* atualiza o seu índice de duas maneiras:

- visita a *web* até ter um certo número de páginas e pára. Quando é preciso atualizar o índice é criado um novo usando o processo descrito. Este tipo de *crawler* chama-se um *crawler* periódico.

- alternativamente o *crawler* continua a visitar páginas mesmo depois de ter atingido o tamanho definido, de modo a atualizar o índice incrementalmente. Neste incre-

mento, o *crawler* substitui as páginas menos importantes por páginas novas, com maior importância. Este tipo de *crawler* é chamado de incremental.

4.1.3 *Wrapper*

Kushmerick[13] definiu *wrapper* como um programa que extrai informação de uma fonte de dados e transforma esses dados para a forma relacional, isto é, de forma a que os mesmos possam ser inseridos numa base de dados. Devido ao sucesso e rápido crescimento das fontes de informação públicas na *Web*, é cada vez mais apelativo extrair dados dessas fontes e disponibilizá-los a utilizadores e aplicações para processamento futuro. Estes dados extraídos de *Web sites* podem ser usados para variadas tarefas, como por exemplo, recolha de informação (*business intelligence*), monitorização de eventos (notícias e mercados financeiros) e comercio eletrónico (comparar compras).

4.2 Tecnologia Usada

Nesta secção vão ser descritas as tecnologias usadas na concretização desta aplicação.

4.2.1 *Bootstrap*

O *Bootstrap* é uma *framework open-source* utilizada no *front-end* para ajudar a desenvolver aplicações *Web* responsivas usando o método *Responsive Web Design* (*RWD*[1]). Para além de conter os vários elementos de *Cascading Style Sheets (CSS)*, o *Bootstrap* apresenta um diversidade de componentes em *JavaScript* e *JQuery*) que ajudam a implementar uma grande variedade funcionalidades, como por exemplo: *tooltip*[2], *menu-dropdown*, *modal*, *carousel*, *slideshow*, etc. Esta *framework* é muito popular entre os *designers*, existindo, assim, várias bibliotecas e *plug-ins* para esta.

4.2.2 *Scalable Vector Graphics (SVG)*

Scalable vector graphics[24] é uma linguagem usada para descrever gráficos de duas dimensões usando *Extensible Markup Language* (*XML*). Permite três tipos de objeto: *vector graphics shapes*[3], imagens e texto. Os objetos feitos usando *SVG* são dinâmicos e interativos, uma vez que o *Document Object Model*(*DOM*) do *SVG* permite animação de gráficos através de *scripts* e é possível definir um conjunto vasto de eventos sobre o objeto *SVG*, como por exemplo *onmouseover* ou *onclick*. Como são objetos escaláveis, podem ser aumentados e diminuídos uniformemente, permitindo que estes sejam visualizados em

[1] Ajuda a programar um *site* de forma a que os elementos que o compõem se adaptem automaticamente à largura do ecrã no qual está a ser visualizado.

[2] É um *pop-pup* que contém uma explicação adicional sobre um determinado elemento.

[3] podem ser, linhas, curvas, etc

qualquer resolução. Sendo os *SVGs* baseados em vetores contém, objetos geométricos, como linhas e curvas em vez de *pixels*, o que faz com que haja uma maior flexibilidade.

Para desenhar o gráfico, os *SVGs* definem várias formas:

- **Retângulo**

  O elemento *rect* define um rectângulo cujos eixos se encontram alinhados com o sistema de coordenadas corrente. Por exemplo:

  <rect x="[No.]" y="[No.]" width="[No.]" height="[No.]" rx="[No.]" ry="[No.]"/> em que *x* e *y* definem as coordenadas do ponto inicial, *width* e *height* correspondem, respectivamente, às dimensões do retângulo e *rx* e *ry* são usadas para desenhar o retângulo com cantos arredondados.

- **Círculo**

  <circle cx="[No.]" cy="[No.]" r="[No.]" /> em quem *cx* e *cy* são as coordenadas *xy* que definem o centro e *r* é o raio.

- **Elipse**

  <ellipse cx="[No.]" cy="[No.]" rx="[No.]" ry="[No.]"/> em que *cx* e *cy* são as coordenadas *xy* que definem o centro e *rx* e *ry* definem o raio.

- **Linha**

  <line x1="[No.]" y1="[No.]" x2="[No.]" y2="[No.]" /> em que (*x1*, *y1*) definem o ponto inicial (*x2*, *y2*) definem o ponto final.

- **Polyline**

  <polyline points"[x1,y1] [x2,y2] ..... [xn,yn]"/> em que *points* representa a lista de pontos que fazem o polígno.

- **Texto**

  <text x="[No.]" y="[No.]" > [Text goes here] </text> em que *x* e *y* são as coordenadas onde começa o texto.

- ***Styling***

  O texto e formas têm mais atributos que são definidos através do elemento *style* por exemplo *style*="*options*"As opções podem ser:

  - *fill:[Color]* – preenche o objeto com a cor dada;
  - *stroke:[Color]* – define cor da linha do objeto;
  - *strokewidth:[No.]* – define a largura da linha;
  - *fontfamily:[Font]* – define o tipo de letra;
  - *fontsize:[No.]* – define o tamanho de letra;
  - *fontstyle:[style]* – define o estilo de letra, ou seja, itálico, bold, etc.

- **Transformações**

  - *matrix(<a> <b><c><d><e><f>)* – define uma transformação em forma de matriz com seis valores;
  - *scale(<sx><sy>)* – aplica a operação *scale* com fator *sx* e *sy*;
  - *rotate(<rotate-angle><cx><cy>)* – aplica uma rotação definida por *rotate-angle* nas coordenadas passadas. Se não forem fornecidas coordenadas é aplicada na origem.;
  - *skewX(<skew-angle>)* – aplica uma transformação *skew* ao longo do eixo dos *x*, ou seja, "empurra" todas as coordenadas do eixo do *x* num ângulo;
  - skewY(<skew-angle>) – aplica uma transformação *skew* ao longo do eixo dos *y*, ou sejam, "empurra" todas as coordenadas do eixo do *y* num ângulo.

4.2.3 Data-Driven Documents (D3.js)

Para construir interfaces web é frequente o recurso a várias ferramentas, principalmente quando se fala de *Web*. As mais comuns são *HyperText Markup Language (HTML)* para o conteúdo da página, *CSS* para a parte estética, *JavaScript* para interação e *SVG* para gráficos vetoriais. Um dos grandes sucessos da *Web* como plataforma é a cooperação entre estas tecnologias. Isto deve-se ao facto de existir uma representação partilhada da página denominada *DOM*. O *DOM* expõe a estrutura hierárquica da página, como parágrafos e tabelas, permitindo referenciá-los e manipulá-los. Ao analisar-se as observações por parte dos utilizadores sobre o Protovis[6], foram identificados três objetivos:

- Compatibilidade
- *Debug*
- Desempenho

Tendo em conta estes pontos, foi criada uma biblioteca em *JavaScript*, denominada *D3.js*[4][7], que é uma linguagem embutida, desenhada para resolver um domínio específico de problemas, que combina elementos de visualização poderosos com uma abordagem orientada

[4] `http://d3js.org/`

a dados para efetuar a manipulação do *DOM*. Com *D3.js*, podem-se associar seletivamente dados a elementos arbitrários do *DOM*, aplicando transformações dinâmicas para gerar e modificar conteúdo. Como tira proveito dos padrões da *Web*, como *HTML*, *SVG* e *CSS*, melhora a expressividade e acessibilidade, enquanto que as transformações oferecem ganhos importantes em termos de desempenho e permitem transições animadas. Através da sua página oferece ainda uma extensa galeria[5] com exemplos que facilitam o uso desta biblioteca assim como uma API[6] que oferece um conjunto de funções que podem ser usadas para criar uma grande variedade de funcionalidades. Uma outra grande vantagem do *D3.js* é que muitas das suas criações são *open-source* e podem ser reutilizadas.

4.2.4 *RESTful Web Services*

Web Services[22] são aplicações cliente-servidor que comunicam através de pedidos *HyperText Transfer Protocol* (*HTTP*). Fornecem uma grande interoperabilidade e extensibilidade, podendo ser combinados para efetuar operações complexas. Dividem-se em duas grandes categorias:

- Baseados em *SOAP*
- *RESTful*

Os *Web Services* baseados em *SOAP* permitem aos sistemas comunicar através de mensagens *SOAP*. Estas mensagens são compostas por *XML*, que são um padrão acessível a qualquer pessoa e aplicação que tenha sido desenhada para o suportar. A figura 4.1 mostra um exemplo de uma mensagem *SOAP*.

```
<SOAPenv:Envelope
      xmlns:SOAPenv="http://schemas.xmlSOAP.org/SOAP/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAPenv:Body>
  <req:getNumberOfArticles xmlns:req="http://daily-moon.com/CMS/">
     <req:category>classifieds</req:category>
  </req:getNumberOfArticles>
 </SOAPenv:Body>
</SOAPenv:Envelope>
```

Figura 4.1: Exemplo de uma mensagem *SOAP*

Estas mensagens são trocadas usando pedidos *HTTP*. O sistema que recebe a mensagem interpreta-a responde de acordo com o esperado devolvendo uma resposta no formato de um nova mensagem *SOAP*.

Esta dissertação irá focar-se nos *RESTful Web Services*, uma vez que foram os utilizados para a concretização da mesma. Os *RESTful Web Services* usam a arquitetura *Representational State Transfer* (*REST*) que especifica restrições, como por exemplo, uma

[5] https://github.com/mbostock/d3/wiki/Gallery
[6] https://github.com/mbostock/d3/wiki/API-Reference

interface uniforme, que uma vez aplicadas a um *Web Service* melhoram o desempenho, escalabilidade e modificabilidade, o que faz com que os serviços funcionem melhor na *Web*. Na arquitetura *REST*, os dados e as funcionalidades são tipicamente *Uniform Resource Identifiers* (*URIs*), ou seja, são *links Web*, que são postos em prática através de operações bem definidas. A arquitetura *REST* restringe a arquitetura cliente-servidor e usa um protocolo de comunicação *stateless*. Nesta arquitetura, o cliente e servidor trocam representações do recurso através de um protocolo e interface padrões.

O facto da arquitetura *REST* ser simples, leve e rápida deve-se aos seguintes princípios:

- **Identificação dos recursos através do *URI*:** um *RESTful Web Service* expõe um conjunto de recursos que são identificados através dos *URIs* que fornecem um espaço de endereçamento global e permitem identificar o serviço.

- **Interface uniforme:** Os recursos são manipulados usando quatro operações:

  - *POST*, que cria um novo recurso;
  - *GET*, que obtem um ou mais recursos;
  - *PUT*, que atualiza um recurso;
  - *DELETE*, que apaga um recurso.

- *Mensagens auto-descritivas:* uma vez que são desacoplados da sua representação permite que possam ser acessados através de vários formatos, como por exemplo, *XML*, *JSON*, etc.

- ***Stateful interactions through hyperlinks*:** interações *stateful* são baseadas no conceito de transferência explícita de estado, para isso existem diversas técnicas para troca de estado, como por exemplo *cookies*. O estado pode ser embutido na mensagem de resposta de modo a apontar para futuros estados válidos da interação.

Os *Web Services* desta dissertação foram desenvolvidos através da *framework Slim*[7], que permite usar os métodos *GET*, *POST*, *PUT* e *DELETE* para obter a informação guardada na base de dados. Esta *framework* verifica qual o tipo de pedido solicitado pelo *URI*, para depois invocar a função que está associada a esse tipo de pedido. A figura 4.2 representa a construção de um pedido *GET* usando a *framework Slim*.

[7] `http://www.slimframework.com/`

```php
<?php
$app = new \Slim\Slim();
$app->get('/books/:id', function ($id) {
    //Show book identified by $id
});
```

Figura 4.2: Exemplo de um pedido *GET*

4.2.5 *Semantria*

Semantria[8][23] é uma *RESTful API*, altamente customizável, que pode ser facilmente treinada para permitir deteção e categorização de entidades. Permite igualmente que cada aspeto da análise de sentimentos possa ser ajustado às necessidades. Esta *API* permite o desenvolvimento em várias linguagens como *C++*, *JAVA*, *PHP*, *.NET*, *Python*, *Ruby* e *JavaScript* através de *Software Development Kits* (*SDKs*). Fornece igualmente bibliotecas com todas as funcionalidades da *API* e guias com boas práticas de implementação.

Processamento de dados

***Semantria* é uma *API* assíncrona ou seja:**

- O conteúdo é enviado e obtido separadamente;
- Não é necessário esperar uma resposta para submeter mais conteúdo para análise;
- O conteúdo pode ser devolvido por uma ordem diferente da que foi submetida;
- Caso o conteúdo seja submetido por diferentes máquinas é possível que uma delas receba conteúdo que foi enviado por outra.

Existem 4 tipos de processamento de dados:

- ***Queue*:** consiste em submeter os documentos para análise. É feito um POST para o servidor que devolve um *HTTP status*.
- ***Request*:** obter o estado de um documento usando o seu *ID*. É feito através de um *GET* e o servidor responde com o estado atual do documento: *queued*, processado ou falha. Se devolver processado devolve igualmente o resultado do processamento. Se devolver *queued* ou falha mostra o erro correspondente.
- ***Retrieve*:** devolve todos os documentos processados. O servidor devolve os resultados de todos os documentos que foram processados.
- ***Cancel*:** retira um documento da *queue*.

[8] https://semantria.com

Existem dois modos de processamento de dados no *Semantria*:

- O processamento em modo detalhado que se divide em várias partes:

  – ***Queue*:** em que os utilizadores colocam os documentos em fila de espera para processamento. Um documento pode ser analisado com uma configuração específica, passando para isso o *config_id* dessa configuração ou com a configuração por defeito, onde nada é passado como *config_id*. *Semantria* garante processamento do documento ao fim de 10 segundos.

  Para um único documento o parâmetro *config_id* deve ser passado com parte do *URL*, e o corpo *POST* em *JSON* deve ter três campos: *ID* do documento, o texto a ser analisado e uma *tag* opcional. Após os documentos serem postos em *queue* cada documento é analisado em separado, sendo que o *Semantria* devolve uma análise para cada documento. A figura 4.3 mostra um pedido *POST* para colocar um documento para análise. Para colocar vários docu-

Figura 4.3: Exemplo de colocação de um documento em *queue*

mentos em *queue*, o utilizador envia vários documentos para análise, sendo que estes são analisados independentemente uns dos outros. O corpo do *POST* em *JSON* deve ter três campos: *ID* documento, o texto a ser analisado e uma *tag* opcional. A figura 4.4 mostra um pedido *POST* para colocar vários documentos para análise.

  – ***Request*:** onde os utilizadores podem pedir análises dos documentos individualmente utilizando para esse fim o *ID* do documento. Se o documento foi processado usando a configuração por defeito não é necessário passar o *config_id* quando se vai pedir o documento, caso contrário, é preciso passar o *config_id* no corpo do pedido *JSON*.

  A figura 4.5 mostra a obtensão da análise de um documento.

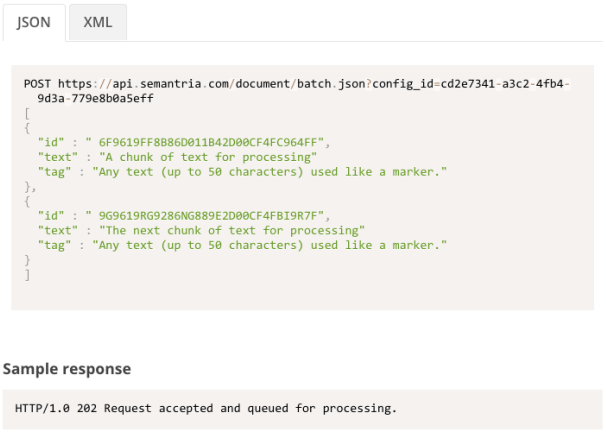

Figura 4.4: Exemplo de colocação de vários documentos em *queue*

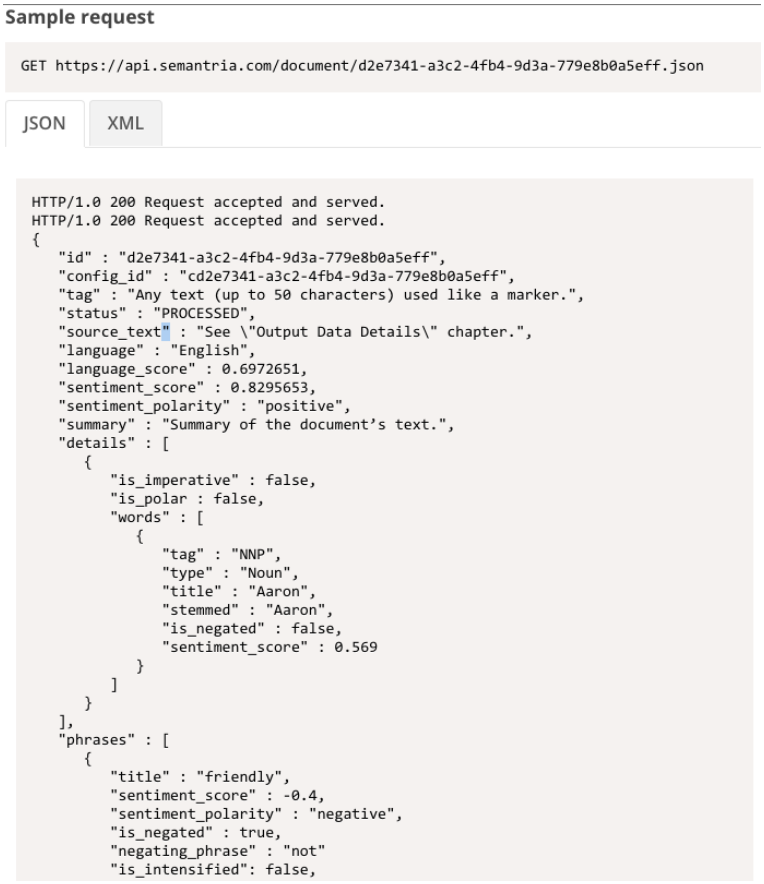

Figura 4.5: Obtensão da análise de um documento

– ***Retrieve*:** onde o utilizador pode pedir todos os documentos que tenham sido processados. Por defeito, o servidor só devolve cem documentos por *batch*. Quando o documento é obtido, o *Semantria* remove-o do sistema. A figura 4.6 mostra um exemplo do pedido necessário para a obtenção das análises de várias documentos.

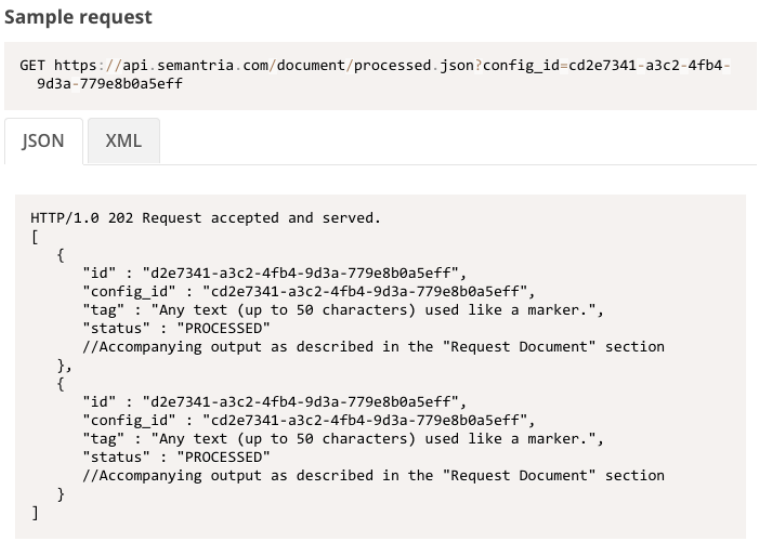

Figura 4.6: Obtensão da análise de vários documentos

- ***Cancel*:** enquanto o documento não tiver sido processado, pode ser removido da *queue*.

- O processamento em modo descoberta que se divide em várias partes:

  - ***Queue*:** para submeter documentos para análise em modo de descoberta é necessário passar um *array* com os *IDs* dos documentos. Neste método, os documentos passados no *array* são analisados em relação uns aos outros e devolve um *output*. O modo descoberta contém um resumo em que mostra os pontos em comum, sentimentos, entidades extraídas, temas e a categoria dos documentos analisados. O corpo do pedido *JSON* deve conter três campos: um *array* com *ID* dos documentos, um *array* com os textos, e uma *tag* opcional. O corpo do pedido, os *IDs* dos documentos, e uma lista dos documentos a serem processados são exigidos para este pedido *POST*. A figura 4.7 mostra um exemplo do pedido necessário para colocar um conjunto documentos para análise em modo descoberta

Figura 4.7: Colocação de *array* de documentos para análise em modo descoberta

– ***Request*:** é pedido ao servidor uma análise específica, caso os documentos tenham sido analisados, devolve a respetiva a análise, caso contrário devolve o estado (em *queue* ou falha).

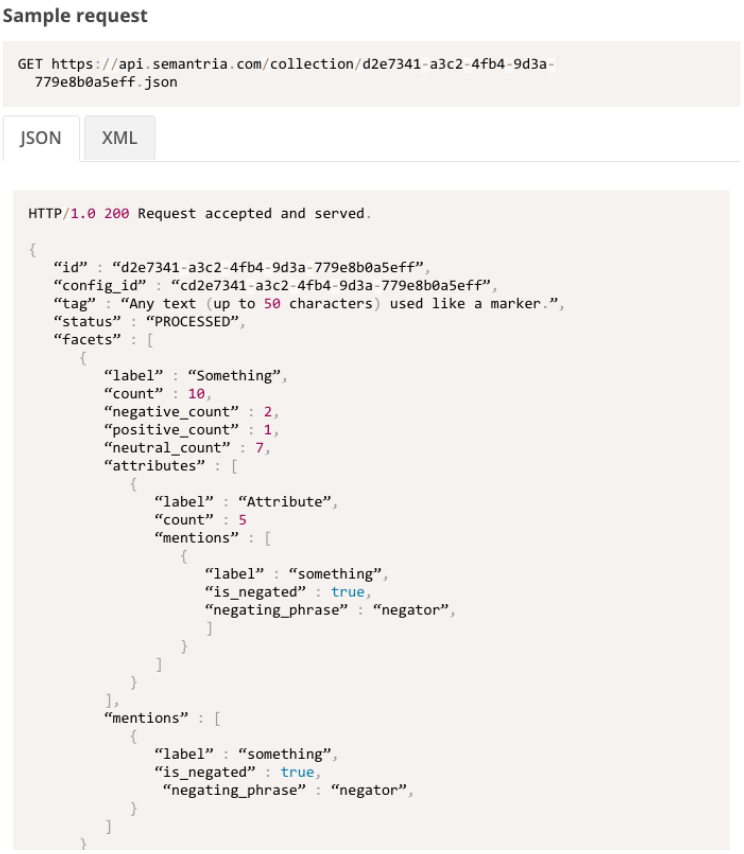

Figura 4.8: Obtensão da análise no modo descoberta

– ***Retrieve*:** Na pesquisa por descoberta o método *retrieve* devolve todas as

análises que estejam completas. A figura 4.9 mostra um exemplo do pedido necessário para obter a análise feita em modo descoberta.

Figura 4.9: Obtensão de todas as análises no modo descoberta

– ***Cancel*:** Remove o documento da *queue*. A figura 4.10 mostra um exemplo de cancelamento de um documento para análise.

Figura 4.10: Cancelar a análise no modo descoberta

4.2.6 *Model-View-Controler (MVC)*

O *MVC*[14] é um padrão de desenho de arquitetura de software para implementar aplicações interativas. A ideia principal é separar os interfaces de utilizador dos dados que representam esse interface. O padrão *MVC* é composto por:

- O *Model* que representa a informação e lógica que muda a informação de acordo com as interações do utilizador.
- A *View* que mostra informação guardada no *Model* ao utilizador e juntamente com o *Controller*, que processa as interações do utilizador, formam o interface da aplicação.
- O *Controller* que é responsável por conter a informação representada na *View*, e igualmente, modificar essa informação de acordo com a interação por parte do utilizador.

O padrão de desenho *MVC* facilita o desenvolvimento e manutenção de aplicações, uma vez que:

- o interface da aplicação pode ser alterado sem necessidade de modificar as estruturas de dados e a lógica de negócio.
- permite manter vários interfaces e diferentes conjuntos de permissões para os utilizadores.

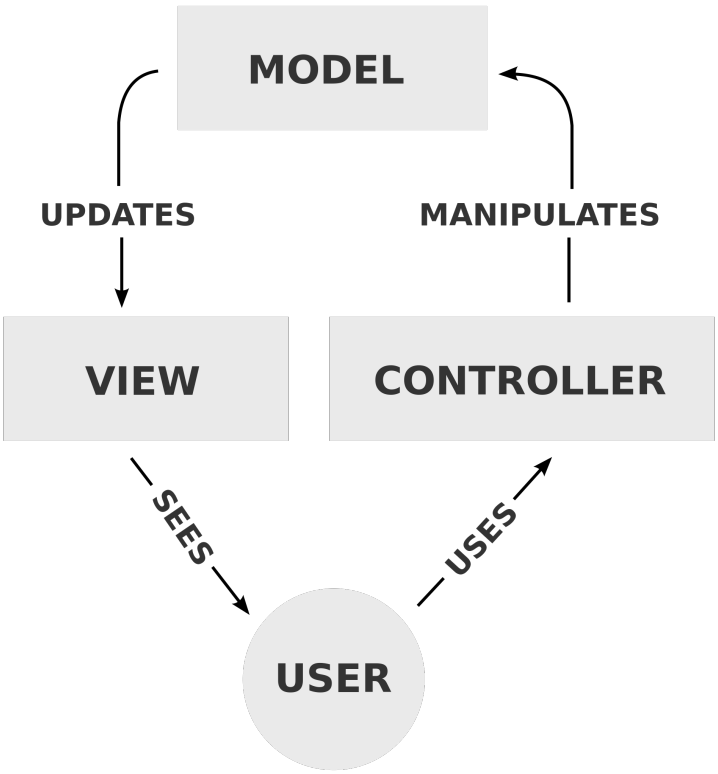

Figura 4.11: Arquitetura *Model-View-Controller*

A figura 4.11 mostra a arquitetura *MVC*. Esta arquitetura é também utilizada para o desenvolvimento de aplicações *Web*. Neste contexto, a *View* é representada por uma página *HTML*, o código que gera os dados dinâmicos para o *HTML* é o *Controller*. O *Model* é tipicamente responsável por representar os dados que se encontram na base de dados. A grande maioria das *frameworks* desenvolvidas, basearam-se na arquitetura *MVC*. Contudo, existem *frameworks* que seguem a arquitetura *MV**, uma vez que não apresentam um *Controller*, isto porque a lógica que opera a aplicação está presente na *View*. Para além disto, este tipo de arquitetura apresenta outros componentes, como por exemplo *Collections* e *Routers*, como é o caso da *framework Backbone.js* .

4.2.7 *Backbone.js*

Backbone.js[9][4], [15], [27], [1], [11] é uma *framework* para desenvolver aplicações *Web* dinâmicas sobre a arquitetura *MV**. É usado para criar aplicações *Web*, representando os dados como *Models* e *Collections*, sendo estes últimos um conjunto de *Models*, que podem ser criados, validados, destruídos e guardados num servidor. Sempre que uma interação do utilizador altera um atributo do *Model* este dispara um evento. As *Views*, que

[9] `http://backbonejs.org/`

representam o estado do *Model*, são notificadas das alterações ao mesmo, sendo renderizadas com a nova informação. A comunicação entre estes elementos é feita através de uma interface *RESTful JSON*. Com esta abordagem, obtém-se um código que não inspeciona e nem depende de todo o *DOM* da aplicação para atualizar manualmente o *HTML*, uma vez que as *Views* estarão sempre atualizadas de acordo com as mudanças verificadas nos *Models*. O *Backbone* é leve e robusto, no entanto depende fortemente da biblioteca *Underscore*[26], que por sua vez fornece diversos recursos para aplicações *Javascript*, como suporte a *templates*, e suporte a recursos de programação funcional. No lado negativo, o motor de *templates* do *Underscore* é muito básico e por isso é necessário incluir *Javascript* como mostrado na figura 4.12.

```javascript
<ul>
    <% _.each(frameworks, function(framework) { %>
        <li title="<%- framework.description %>">
            <%- framework.name %>
        </li>
    <% }); %>
</ul>
```

Figura 4.12: Exemplo de utilização do *Underscore*

O *backbone* permite separar a lógica do negócio do interface de utilizador. Isso é conseguido através de:

- *Models* que gerem os dados da aplicação e envia notificações sempre que os mesmos são alterados. Devem poder ser passados ao longo da aplicação para onde os dados são precisos.

- *Views* que fazem parte do interface do utilizador. Normalmente processam dados de um *Model* ou conjunto de *Models*, podendo também existir sem depender dos dados de um *Model*. Monitorizam alterações ao *Models* e atualizam-se para mostrar essa mudança. A comunicação entre *Views* e *Models* está representada na figura 4.13.

Figura 4.13: Exemplo da comunicação entre *Models* e *Views*

- *Collections* que ajudam a lidar com um conjunto de *models* que estão relacionados entre si, tratando de carregar e guardar novos *models* no servidor. Fornece igualmente funções que permitem efetuar agregações e cálculos sobre uma lista de *models*. Uma *collection* está igualmente atenta aos eventos que acontecem nos eventos dentro de si, permitindo, num único local, escutar alterações que aconteçam nos *models*. Esta comunicação é ilustrada na figura 4.14.

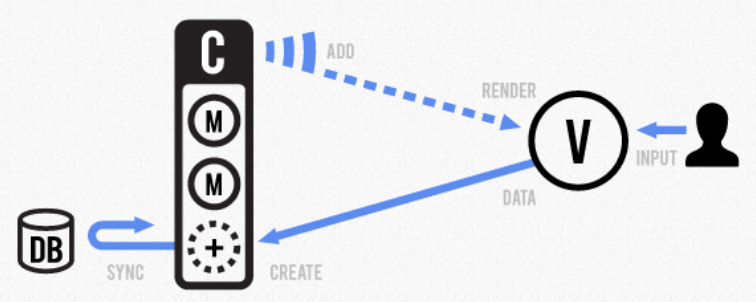

Figura 4.14: Arquitetura *Exemplo da comunicação entre Collections e Views*

Renderização das *Views*

Uma *View* liga um *Model* à sua representação visual no *DOM*. A *View* apresenta os dados do *Model* associado, capturam o *input* do utilizador e enviam de volta para o *Model*. Existem duas formas de uma *View* ser renderizada. A primeira é quando um utilizador realiza uma ação e espera que a mesma *View* seja novamente renderizada, a segunda é no caso de um utilizador clicar, por exemplo num *link* e os dados obtidos do servidor são representados por outra *View*, como exemplificado na figura 4.15.

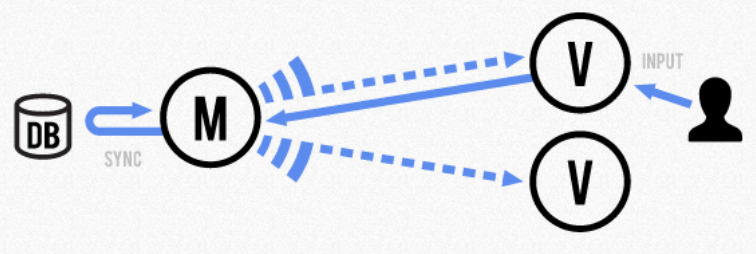

Figura 4.15: Arquitetura *Exemplo de Renderização de Views*

Routing com *URL*

Nas aplicações *Web* pretende-se fornecer *URLs* para localizações dentro da aplicação que sejam *linkables*, marcáveis e partilháveis. O *Router* consegue detetar mudanças no *URL* e pode dizer à aplicação exatamente onde se encontra. Por exemplo, se um utilizador clicar num *link* e este contenha a *hashtag books* (*#books*), é verificado a que *Router* pertence o "*#books*". Este exemplo está ilustrado na figura 4.16.

4.2.8 *Angular.js*

Esta é uma *framework*[3], [11] estruturada para criar páginas *Web* dinâmicas. Apresenta um grande nível de abstração o que simplifica o desenvolvimento de aplicações e por

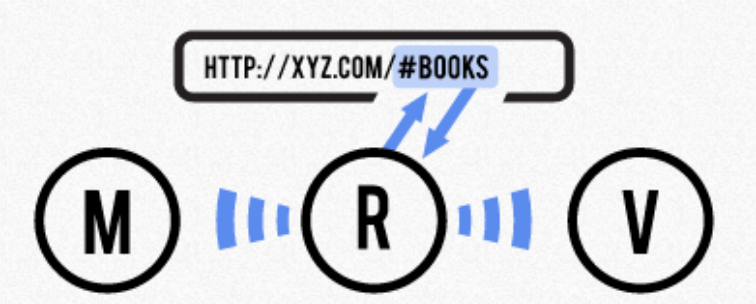

Figura 4.16: Exemplo do funcionamento do *Routing*

essa razão pode tornar-se pouco flexível. Normalmente, ao desenvolver-se aplicações dinâmicas, existe uma resistência entre estas e os documentos estáticos. Para resolver isso, o *Angular.js* utiliza:

- *Data-binding*, *Directives*, criação e validação de *forms HTML*, *Routing*, *deep-linking*, reutilização de componentes *HTML* e injeção de dependências.

- Histórico de testabilidade através de testes unitários, testes ponto a ponto, *mocks* e testes automatizados.

O AngularJs é composto principalmente por:

- ***Data binding***: O *template*[10] é compilado, o que gera uma *View*. Se a *View* for alterada, essas alterações são imediatamente propagadas para o *Model* e vice-versa. Como a *View* é apenas uma projeção do *Model* e o *Controller* está separado da *View*, este não conhece o estado do *Model*. Esta separação permite testar o *Controller* isoladamente. Este funcionamento está exemplificado na figura 4.17.

[10]que é codigo *HTML* descompilado juntamente com diretivas do *Angular*

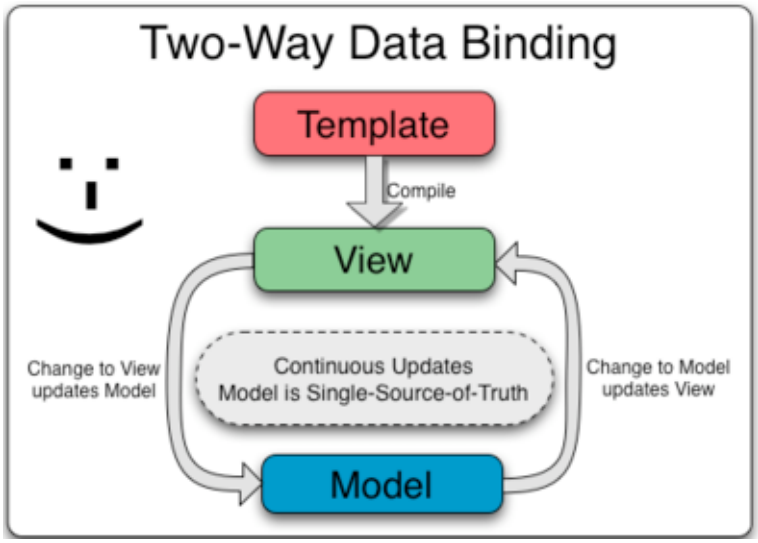

Figura 4.17: Exemplo de *Data Binding* em *Angular.js*

- **Expressões**: A fim de criar *Views* numa aplicação, o *Angular.js* permite executar expressões diretamente dentro das páginas HTML, como exemplificado na figura 4.18.

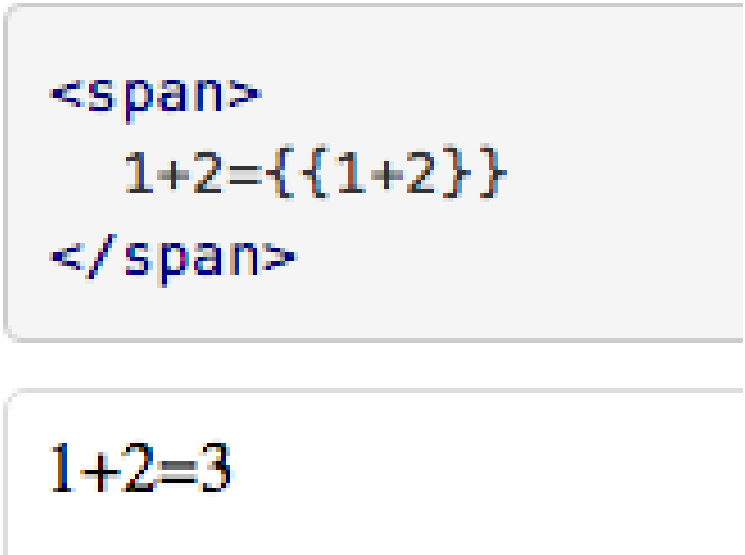

Figura 4.18: Exemplo da sintaxe das expressões em *Angular.js*

- ***Scope***: O *Scope* é o objeto que refere o *Model* da aplicação. É usado como contexto de execução para expressões, observa-as e propaga eventos. O *Scope* tem uma *API* para observar mutações no *Model* e outra para propagar essas mudanças para a *View*. O funcionamento do *Scope* é exemplificado na figura 4.19.

```
angular.module('scopeExample', [])
.controller('MyController', ['$scope', function($scope) {
  $scope.username = 'World';

  $scope.sayHello = function() {
    $scope.greeting = 'Hello ' + $scope.username + '!';
  };
}]);
```

Figura 4.19: Exemplo da sintaxe do *Scope* em *Angular.js*

- ***Directives***: As *Directives* são elementos incorporados e definidos no *HTML*, de forma a que a manipulação do *DOM* seja mais transparente e/ou para adicionar novos comportamentos às *tags* existentes. Um exemplo de uma *Directive* é o "*ngrepeat*", que é utilizada para fazer iterações sobre um *array* e renderiza um *template* para cada elemento do *array*. As *Directives* são marcadores num elemento do *DOM*[11], que diz ao compilador de HTML do *Angular.js*, para vincular um comportamente específico ou mesmo para transformar esse elemento do *DOM*. O *Angular.js* vem com um conjunto de *Directives*, como por exemplo, *ngBind*, *ngModel*, and *ngClass*. A figura 4.20 representa um exemplo de utilização de *Directives*.

```
<div ng-controller="Controller">
  Hello <input ng-model='name'> <hr/>
  <span ng-bind="name"></span> <br/>
  <span ng:bind="name"></span> <br/>
  <span ng_bind="name"></span> <br/>
  <span data-ng-bind="name"></span> <br/>
  <span x-ng-bind="name"></span> <br/>
</div>
```

Figura 4.20: Exemplo da sintaxe das *Directives* em *Angular.js*

- ***Controller***: O *Controller* é definido pelo construtor de uma função de *JavaScript* e é usado para aumentar o *Scope* da aplicação. Quando um *Controller* é associado ao *DOM* através da *Directive ng-controller*, o *Angular.js* irá criar um novo *Controller* usando o contrutor apropriado. É igualmente criado um *Scope* filho, que é disponibilizado como um parâmetro do construtor do *Controller*, através da variável *$scope*. É usado para definir o estado inicial e adicionar comportamento ao *$scope*, como se pode verificar na figura 4.21.

[11] como um atributo, nome, comentario ou classe de CSS

```
var myApp = angular.module('myApp',[]);

myApp.controller('DoubleController', ['$scope', function($scope) {
  $scope.double = function(value) { return value * 2; };
}]);
```

Figura 4.21: Exemplo da sintaxe do *Controller* em *Angular.js*

- ***Modules***: Os *Modules* são componentes que inicializam e encapsulam os *Controllers*, *Directives*, *Services* e *Routes* e são definidos pela *Directive* "*ng-app*". A figura 4.22

```
<div ng-app="myApp">
  <div>
    {{ 'World' | greet }}
  </div>
</div>
```

Hello, World!

Figura 4.22: Exemplo da sintaxe do *Module* em *Angular.js*

- ***Services***: *Services* são objetos que são ligados através de injeção de dependências e são usados para organizar e partilhar código pela aplicação. São *lazy instaciated*, ou seja, apenas são instanciados quando um componente da aplicação depende disso; e *singletons*, ou seja, cada componente dependente de um serviço recebe uma referência para uma única instância gerada pelo serviço. A figura 4.23 mostra a utilização de um *Service*.

```
<div id="simple" ng-controller="MyController">
  <p>Let's try this simple notify service, injected into the controller...</p>
  <input ng-init="message='test'" ng-model="message" >
  <button ng-click="callNotify(message);">NOTIFY</button>
  <p>(you have to click 3 times to see an alert)</p>
</div>
```

Figura 4.23: Exemplo da sintaxe do *Service* em *Angular.js*

- ***Routers***: Esta componente permite criar diferentes *URLs* para diferentes páginas numa aplicação. Providenciam *URLs linkables*, marcáveis e partilháveis dentro da

própria aplicação. A definição de um *Router* está representada na figura 4.24.

```
angularJSApp.config(['$routeProvider', function ($routeProvider){
    $routeProvider.when('/users', {
        templateUrl: 'views/users.html',
        controller: 'UsersCtrl'
    }).otherwise({
        templateUrl: 'views/error.html'
    });
}]);
```

Figura 4.24: Exemplo da sintaxe do Router *em Angular.js*

4.2.9 *BackBone.js* vs. *Angular.js*

Estas duas *frameworks*[11, 12] têm muitos aspetos em comum: ambas são *open source*, e tentam resolver o problema de desenvolver *SPA* usando o design *MVC* e *MV**. Ambas apresentam os conceitos de *Models*, *Views*, *Events* e *Routing*. O *Angular.js* tem uma comunidade maior e em maior crescimento, mais projetos com estrela no *GitHub* e mais perguntas no *StackOverflow* que o *Backbone.js*[25]. Por outro lado, o *Backbone* é mais rápido a carregar páginas que o *Angular.js*, é mais flexível que o último e tem uma curva de aprendizagem menor. Por estas razões, no desenvolvimento da aplicação *Web* foi decidido optar por usar a *framework Backbone* em detrimento do *Angular.js*.

4.3 Arquitetura

Para alcançar os objetivos definidos para esta dissertação na secção 1.2, foi desenvolvida a aplicação *Rankores*. O primeiro passo foi desenhar a arquitetura do sistema de forma a mostrar como a aplicação está organizada, quais os seus componentes e como é que eles estão divididos.

A figura 4.25 representa a arquitetura da aplicação.

No *back-end* colocou-se a base de dados usada para armazenar os dados recolhidos pelos restantes componentes do *back-end*, *wrappers* e os clientes das redes sociais, bem como, os *Web Services* responsáveis pela comunicação entre os restantes componentes da aplicação e a base de dados. A secção 4.4, fornece uma explicação mais detalhada sobre o funcionamento destes componentes.

O componente de *front-end* é responsável por apresentar o *interface* da aplicação ao utilizador, bem como, replicar as interações que este efetua no *interface* da aplicação nos restantes componentes e mostrar os resultados que estas interações têm na aplicação. Uma descrição mais detalhada de como estes componentes foram implementados será fornecida nas secções, 4.6 e 4.7.

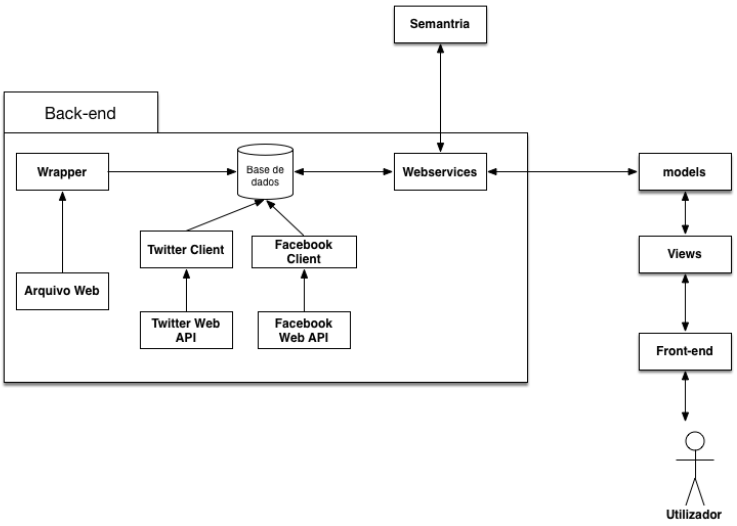

Figura 4.25: Arquitetura da aplicação

4.4 *Back-end*

Nesta secção é descrita a implementação dos componentes que compõem o *back-end* da aplicação.

4.4.1 Recolha de informação

Na parte de *back-end* tirou-se proveito do conceito exposto na secção 4.1.3, para criar os *wrappers* responsáveis pela recolha da informação necessária para enriquecer a base de dados de modo a existirem dados sobre os quais se pudesse trabalhar. Foram desenvolvidos *wrappers* com as seguintes responsabilidades:

- *wrapper* responsável pela recolha de jogadores,
- *wrapper* responsável pela recolha de equipas,
- *wrapper* responsável pela recolha das ligas,
- *wrapper* responsável pela recolha de notícias desportivas de várias fontes (Record [12], A bola [13], MaisFutebol [14] e Relvado [15]).

Os *wrappers* desenvolvidos leêm *URLs* de um ficheiro e para cada um deles obtêm-se o conteúdo *HTML* da página. Esse conteúdo *HTML* é carregado para um objeto to tipo *DOM*, que vai ser usado para percorrer a estrutura da página, tirando partido do

[12] `http://www.record.xl.pt`
[13] `http://www.abola.pt`
[14] `http://www.maisfutebol.iol.pt`
[15] `http://relvado.aeiou.pt`

DOMXPath[16], até encontrarem a informação que pretendem, guardando a mesma na base de dados.

Por exemplo, para uma notícia [17], é recolhido da mesma, o título, descrição, data em que foi publicada, autor da notícia e o corpo da notícia. Caso a notícia não tenha algum destes campos, este não é introduzido na base de dados, ficando o campo correspondente vazio. A figura 4.26, mostra a localização destes campos na página de uma notícia. As figuras 4.27 e A figura 4.28 mostram como é feita a recolha destes campos pelo *wrapper*.

Figura 4.26: Campos recolhidos pelo *wrapper* na página de uma notícia

Para fazer face ao objetivo de mostrar a reação das pessoas às notícias publicadas nas redes sociais, que se encontra descrito na secção 1.2, foram criados programas que

[16] `http://php.net/manual/en/domxpath.query.php`

[17] `http://www.record.xl.pt/futebol/nacional/1a-liga/sporting/detalhe/leao-quer-saber-fundamentos-do-castigo-a-joao-mario-973763.html`

```
while (!feof($file)){
    $url = fgets($file);
    $url = str_replace("\n", "", $url);
    $url = str_replace("\r", "", $url);
    echo "Crawlar ->" . $url . "\n";

    $html = file_get_html($url);
    $dom = new DOMDocument();
    @$dom->loadHTML($html);
    $xpath = new DOMXPath($dom);
    getNews($xpath, $aBola, $con);
}
```

Figura 4.27: Exemplo da recolha feita pelo *wrapper* - Passo 1

```
$titulo = utf8_decode( $xpath->query("//*[@id='body_ver_pnlNoticia']/div/div[4]")->item(0)->nodeValue);
$data_noticia = $xpath->query("//*[@id='body_ver_pnlNoticia']/div/div[5]")->item(0)->nodeValue;
```

Figura 4.28: Exemplo da recolha feita pelo *wrapper* - Passo 2

fazem a ligação às redes sociais e recolhem as noticias partilhadas pelas páginas dos jornais desportivos. Isto foi concretizado pelos componentes "*Facebook Client*"e "*Twitter Client*", que recolhem as notícias partilhadas nas respetivas redes sociais, juntamente com os comentários e gostos no caso do *Facebook* e *retweet* no caso do *Twitter*

Para se desenvolver estes componentes tirou-se partido das *APIs* do *Facebook*[18] e *Twitter*[19], respetivamente.

No caso do *Facebook*, para usarmos a sua *API* é necessário criar uma aplicação para obter um *App ID* e um *App Secret*, que juntos geram um *Auth Token* que é usado para a aplicação se autenticar perante a *API*. A figura 4.29, mostra como é criado esse *token*. Usando esse *token* como forma de autenticação, a aplicação pode agora efetuar pedidos à

```
//App Info, needed for Auth
$app_id="787556588027224";
$app_secret="0bb4a1a664258047031c318be754e04a";

//Retrieve auth token
$authToken = fetchUrl("https://graph.facebook.com/oauth/access_token?grant_type=client_credentials&client_id={$app_id}&client_secret={$app_secret}");
```

Figura 4.29: Autenticação na *API* do *Facebook*

API. No caso específico desta dissertação os pedidos foram feitos sobre o *feed* das páginas de *Facebook* dos jornais desportivos portugueses. A resposta desses pedidos é devolvida em formato *JSON*, sendo processada para se obter a informação pertinente ao âmbito da aplicação. A figura 4.30, mostra um exemplo do processo descrito.

```
$json_object =  fetchUrl("https://graph.facebook.com/{$profile_id}/feed?{$authToken}");
$feedArray = json_decode($json_object);
getNewsFields($feedArray, $profile_id);
```

Figura 4.30: Exemplo de um pedido à *API* do *Facebook*

[18] https://developers.facebook.com/
[19] https://dev.twitter.com/rest/public

Para usar a *API* do *Twitter* o processo é semelhante ao usado no *Facebook*, na medida em que também é necessário criar uma aplicação [20] à qual é atribuída uma *Consumer Key*, um *Consumer Secret*, um *Access Token* e um *Access Token Secret*. Estes irão permitir à aplicação autenticar-se perante a *API* e efetuar pedidos à mesma.

No âmbito desta dissertação, os pedidos feito à *API*, contemplam obter os *tweets* nos quais os jornais desportivos partilham notícias no *Twitter*. A resposta aos pedidos vem, como sucede no *Facebook*, em formato *json*, que posteriormente é processado para retirar a informação relevante. A figura 4.31 representa um pedido feito usando a *API* do *Twitter*.

```php
/** Set access tokens here - see: https://dev.twitter.com/apps/ **/
$settings = array(
    'oauth_access_token' => "1376914609-kCxoUAxnuiloJbUpBQubBeT6IB1KXHvwAr20Nlw",
    'oauth_access_token_secret' => "sUKnmsHzelYAMBM69cmkJAF25Emf9OrlUeANOsDWkqeS3",
    'consumer_key' => "5Ovcacv9Pm8ha4HOMb0QTlZJ3",
    'consumer_secret' => "qlM174uKckMhC5raosFBFW6TuxzWIV98fJaiW8LoZT7Tbi1LR6"
);

$url = 'https://api.twitter.com/1.1/statuses/user_timeline.json';
/* $getfield = '?screen_name=@abolapt'; */
/* $getfield = '?screen_name=@Record_Portugal'; */
/* $getfield = '?screen_name=@maisfutebol'; */
/* $getfield = '?screen_name=@ojogo'; */
$getfield = '?screen_name=@relvado_pt';
$requestMethod = 'GET';
$twitter = new TwitterAPIExchange($settings);
$response = json_decode($twitter->setGetfield($getfield)
            ->buildOauth($url, $requestMethod)
            ->performRequest(), true) ;
```

Figura 4.31: Exemplo pedido à *API* do *Twitter*

[20] https://apps.twitter.com/

4.4.2 Base de Dados

Um base de dados é um componente essencial de qualquer sistema ou aplicação, uma vez que é nela que são guardados os dados recolhidos pelo *back-end*, assim como os dados resultantes da validação das notícias por parte dos utilizadores. Para representar a base de dados responsável por armazenar os dados recolhidos pelos componentes da secção 4.4.1, foi usado o modelo conceptual, Entidade-Associação, ilustrado na figura 4.32.

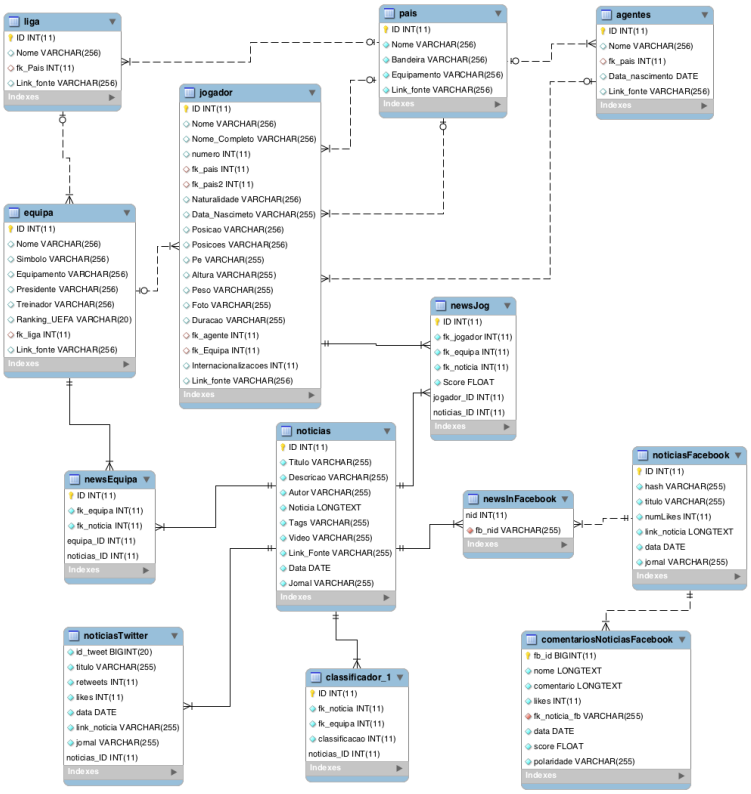

Figura 4.32: Arquitetura da base de dados

A base de dados da aplicação é formada por:

- Tabela "Liga"que guarda informações sobre uma liga.
- Tabela "País"que guarda informação sobre um país.
- Tabela "Agentes"que guarda informação sobre agentes desportivos.
- Tabela "Jogador"que guarda informação sobre jogadores de futebol.
- Tabela "Equipa"que guarda informação sobre equipas de futebol.
- Tabela "Notícias"que guarda informação sobre notícias de futebol.
- Tabela "*NewsJog*"que relaciona as notícias com jogadores, ou seja, esta tabela dá informação se um determinado jogador tem notícias sobre si e quais.

- Tabela "*NewsEquipa*"que relaciona as notícias com equipas, ou seja, esta tabela dá informação se uma determinada equipa tem notícias sobre si e quais.
- Tabela "NotíciasFacebook"que guarda informação sobre as notícias partilhadas pelos jornais na rede social *Facebook*.
- Tabela "ComentáriosNotíciasFacebook"que guarda a informação sobre os comentários sobre notícias partilhadas no *Facebook*.
- Tabela "NotíciasTwitter"que guarda informação sobre as notícias partilhadas na rede social *Twitter*.
- Tabela "NoticiasInFacebook"que relaciona as notícias que são partilhadas na rede social *Facebook* com as que são partilhadas nas páginas *Web* dos jornais.
- Tabela "NoticiasInTwitter"que relaciona as notícias que são partilhadas na rede social *Twitter* com as que são partilhadas nas páginas *Web* dos jornais.

4.5 Estrutura da Aplicação

A *framework Backbone.js* permite criar uma estrutura de forma a facilitar a manutenção do código e a atualização da interface da aplicação. A figura 4.33 representa essa organização.

Nesta estrutura os ficheiros e diretorias mais importantes são, ***css***, ***index.html***, ***js***, ***app.js***, ***models***, ***views***, ***tpl*** e ***Web Services***. Na pasta ***css*** encontram-se os ficheiros responsáveis pelo estilo da aplicação, contém elementos da *framework Bootstrap*.

O ficheiro ***index.html*** é usado para carregar o estilo da páginas e *scripts* necessários para o normal funcionamento da aplicação. Define igualmente a estrutura inicial do *DOM* na qual as *Views* vão ser carregadas.

A diretoria ***js*** contém os elementos responsáveis pelo funcionamento da aplicação, os *Models* que fazem a ligação à base de dados e e tiram partido dos ***Seb Services*** para ler e escrever dados na base de dados, as *Views* que em conjunto com os seus *templates* são usadas para representar os dados da aplicação e implementam as funcionalidades da aplicação e o ficheiro ***app.js***. Este ficheiro é o ficheiro mais importante da aplicação, visto que é aquele que acarreta maiores responsabilidades, dado que neste ficheiro são carregados todos os *templates* da aplicação, sobre os quais é construído o esqueleto da aplicação e contém os *Routers* que são responsáveis pela navegação entre as páginas da aplicação.[21]

Por fim, mas não menos relevante, temos a pasta ***tpl***, onde são guardados os *templates* onde é feito o carregamento das *Views*.

[21] *Home*, Notícias, Votar, de Jogadores, Concretizadas, Não Concretizadas, Estatísticas e Redes Sociais.

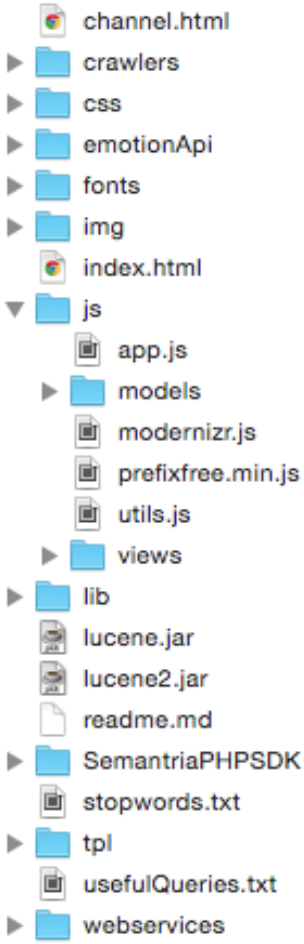

Figura 4.33: Estrutura da aplicação

4.6 *Models* e *Collections*

Os *Models*, como referido anteriormente na secção 4.2.7, são os componentes responsáveis por gerirem os dados da aplicação, permitindo ler e modificar os mesmos consoante as interações do utilizador da aplicação. As *Collections* são um conjunto de *Models*. De modo a concretizarem o seu propósito, os *Models* e *Collections* precisam de comunicar com a base de dados, comunicação essa que é assegurada por um conjunto de *Web Services*. Ambos possuem três tipos de funções para efetuar comunicação com a base de dados e manipulação dos respetivos dados contidos nesta:

1. ***save*** - este método permite fazer um pedido *POST* ou *PUT* para guardar ou atualizar,

2. ***fetch*** - este método permite fazer um pedido *GET* para obter dados da base de dados.

3. ***destroy*** - este método permite fazer um pedido *DELETE* para eliminar dados da base de dados.

De seguida são apresentados dois exemplos de *Models* usado pela aplicação.

4.6.1 *Model* ComentáriosNews

Este *Model* tem como propósito a obtenção de comentários sobre uma dada notícia. Para tal, possui um construtor que recebe o *ID* da notícia pretendida, retornando os comentários existentes para essa notícia. A figura 4.34 mostra com é criado o *Model* "ComentáriosNews".

```
fb.models.ComentariosNews = Backbone.Collection.extend({

    initialize: function(options){
        console.log (options.id);
        this.id = options.id;
    },

    url: function(){
        console.log(this.id);
        return 'http://www.lasige.di.fc.ul.pt/webtools/Rankores/webservices/comentariosNewsFb/' + this.id;
    }
});
```

Figura 4.34: *Model* responsável por obter os comentários de uma notícia

Uma vez criada uma instância do tipo "ComentáriosNews", esta é usada na *View* onde os dados por ele devolvidos são necessários e passados para o *template* dessa *View* através da função *fetch*. A figura 4.35 fornece um exemplo para o *Model* "ComentáriosNews".

```
var comentarios = new fb.models.ComentariosNews({id: hash});
comentarios.fetch({
    success: function(comentarios){
        console.log(comentarios);
        $('#coments').html(new fb.views.ComentariosNoticia({comentarios : comentarios.models}).el);
        return that;
    }
})
```

Figura 4.35: Inicialização do *Model* responsável por obter os comentários de uma notícia

4.6.2 Model Voto

O *Model* "Voto", é responsável por inserir na base de dados a validação feita pelo utilizador sobre uma dada notícia. Para isso, o *Model* contém que um *URL* responsável por fazer um POST na base de dados.

```
fb.models.Voto = Backbone.Model.extend({
    urlRoot: 'http://www.lasige.di.fc.ul.pt/webtools/Rankores/webservices/postVote'
});
```

Figura 4.36: Inicialização do *Model* responsável por inserir a validação do utilizador sobre uma notícia

Cria-se então uma instância do tipo "Voto", que recebe como parâmetro a opção escolhida pelo utilizador para validar a notícia e o código que identifica essa notícia na base de dados. Os dados são então gravados na base de dados através da função *save*.

```
var votar = new fb.models.Voto({valor: voto, noticia: codigo, equipas: equipas.models});
votar.save(null, {
    success: function (){
        console.log("FUNCOU");
        router.navigate('vote', {trigger: true}); */
        window.history.back();
    }
})
```

Figura 4.37: Inicialização do *Model* "Voto"para guardar a validação feita sobre uma notícia

4.7 *Views*

Nesta secção, será explicado o processo de renderização de *Views* na aplicação desenvolvida.

4.7.1 Renderização das *Views*

Vimos na secção 4.5 que o ficheiro ***app.js*** é o ficheiro mais importante da aplicação, uma vez que é ele responsável por iniciar a aplicação com os *templates* associados a cada *View* bem como o *Router* que permite a navegação entre as páginas da aplicação. Quando um utilizador acede à aplicação escrevendo o *URL* no *browser*, o ficheiro ***app.js***, vai carregar a *View shell.js* associada ao *template shell.html*, que é renderizada na *div* com *id=shell*, como é ilustrado pelas figuras 4.38 e 4.39.

```
<html lang="en">
<head>...</head>

<body>

<div class="navbar-wrapper" id="shell"></div>

</body>
```

Figura 4.38: Ficheiro *index.html*

Após o carregamento desta *View*, o ficheiro ***app.js*** determina através do *Router* presente nele, qual a *View* que é necessário carregar para responder ao *URL* introduzido pelo utilizador. A *View* representada na figura 4.40 é então renderizada na *div* com *id=myCarousel* presente no *template shell.html*, ilustrado na figura 4.41.

```
$(document).on('ready', function() {
    // fb.user = new fb.models.Person(); // Holds the authenticated Facebook user
    // Load HTML templates for the app
    fb.templateLoader.load(['shell', 'welcome', 'marquee', 'perfilNoticiaFb',
     'comentariosNoticia', 'social', 'timeline', 'noticiasPeriodo',
     'filtroEquipas', 'newsEquipa', 'votar', 'simbolos', 'news-vote',
     'stats', 'rumores', 'concretizadas', 'naoConcretizadas',
     'jornais', 'newsJogadores', 'jogadoresEquipa', 'jogadoresNoticia'], function () {
        fb.shell = new fb.views.Shell({el: "#shell"});
        fb.router = new fb.Router();
        Backbone.history.start();
    });
});
```

Figura 4.39: Renderização da *View shell.js*

```
fb.views.Shell = Backbone.View.extend({

    initialize: function () {
        this.template = _.template(fb.templateLoader.get('shell'));
        this.render();
    },

    events: {...},

    popUp : function(ev){...},

    popDown : function(ev){...},

    render: function () {
        //loading...
        $('#spinner').fadeIn('fast');
        var lastNews = new fb.models.NewsRedesSociais();
        lastNews.fetch({
            success:function(lastNews){
                //loading...
                $('#spinner').stop().fadeOut('fast');
                $('#myCarousel').html(new fb.views.Marquee({
                lastNews : lastNews.models}).el);
            }
        })
        this.$el.html(this.template());
        return this;
    }

});
```

Figura 4.40: Renderização da *View* da página inicial

```
<!-- <div class="navbar-wrapper"> -->
    <div class="container">
        <nav class="navbar navbar-inverse navbar-static-top">...</nav>
    </div>
<!-- </div> -->

<div id="myCarousel" class="carousel slide" data-ride="carousel">

</div>

<div class="container marketing" id="content">
```

Figura 4.41: Renderização da *View shell.js*

4.8 Identificação de entidades nas notícias

Nesta secção irá ser explicado como são detetadas as entidades relevantes para esta dissertação, ou seja, como é feita a deteção de jogadores e equipas numa notícia.

Para cada jogador existente na base de dados, é procurado o nome do mesmo em todas as notícias da base de dados. Se o nome do jogador existe vamos procurar pela equipa a que ele pertence na mesma notícia, caso ambos existam, dizemos que o jogador ocorre na notícia.

Para as equipas consideramos que uma equipa existe na notícia se um esta ocorre na notícia simultaneamente com um jogador seu.

4.9 Análise de Sentimentos

A análise de sentimentos na aplicação é feita apenas para as notícias provenientes do *Facebook*, uma vez que nesta rede social os utilizadores podem comentar a publicação diretamente no jornal que partilhou a notícia, o que não se verifica no *Twitter*.

Para isso foram recolhidos diariamente notícias dos principais jornais desportivos e respetivos comentários que foram guardados posteriormente na base de dados. De seguida foi criado um ficheiro *PHP* no qual se vai buscar esses comentários através de um *Web Service* que comunica com a base de dados. Os comentários devolvidos por esse *Web Service* são então passados pelo mecanismo de classificação do *Semantria*, que para cada comentário devolve um sumário com o sentimento ilustrado pelo comentário.

Capítulo 5

Resultados

Concluído o desenvolvimento da aplicação foi necessário testa-la. Para isso efetuaram-se testes de usabilidade presenciais com utilizadores onde, foi apresentada uma lista de tarefas a realizar na aplicação. Esta secção mostra os resultados desses testes, bem como a respetiva análise dos mesmos.

5.1 Testes Usabilidade

A usabilidade é um atributo de qualidade que avalia a facilidade de uso do interface. Deste modo os testes de usabilidade são utilizados para avaliar interfaces de utilizador de acordo com este atributo.

A usabilidade é definida por cinco componentes de qualidade[20]:

- **Aprendizagem:** quão fácil é para os utilizadores efetuarem tarefas na primeira vez que interagem com o interface.
- **Eficiência:** depois dos utilizadores "conhecerem"o interface quão rápido conseguem efetuar tarefas no mesmo.
- **Memória:** Após um período em que não utilizou o interface, quão fácil é para ele realizar tarefas.
- **Erros:** Quantos erros cometem os utilizadores, quão graves são esses erros e se recuperam facilmente dos mesmos.
- **Satisfação:** se os utilizadores ficaram satisfeitos com a aplicação.

Adicionalmente quando se desenvolve interfaces direcionados ao utilizador deve-se ter em conta as dez heurísticas definidas por *Nielsen*[19]:

1. **Visibilidade do estado do sistema:** Os utilizadores devem sempre ser informados do que se está a passar, através de *feedback* apropriado dentro de um tempo razoável.

2. **Correspondência entre o sistema e a vida real:** O sistema deve "falar"a linguagem do utilizador, através de frases, termos e expressões que sejam familiares aos utilizadores. Deve seguir convenções do mundo real devendo a informação aparecer de forma natural e lógica.

3. **Liberdade e controlo do utilizador:** O sistema deve suportar *undo* e *redo* para resolver escolhas erradas por parte dos utilizadores, por exemplo, os utilizadores costumam escolher funções de sistema por engano e precisam de uma maneira sair de lá, sem terem de passar por um diálogo longo.

4. **Consistência e padrões:** O sistema deve ser consistente e seguir os padrões definidos.

5. **Prevenção de erros:** Melhor que uma boa mensagem de erros é prevenir situações que levem à ocorrência de erros, através de um design cuidadoso eliminando situações propensas a erros ou apresentar aos utilizadores a opção de confirmarem a sua ação antes desta ser irreversível.

6. **Reconhecimento em vez de recordar:** Minimizar o uso de memória por parte do utilizador, tornando objetos, ações e opções visíveis. As instruções do sistema devem estar facilmente acessíveis.

7. **Flexibilidade e eficiência de uso:** O sistema deve fornecer aceleradores (tais como teclas de atalho), que passam despercebidos a utilizadores inexperientes, mas que melhoram a eficiência da interação do utilizador experiente. O sistema deve estar preparado para albergar tanto utilizadores experientes como inexperientes.

8. **Estética e design minimalista:** O sistema só deve mostrar informação relevante para a utilização do mesmo.

9. **Ajudar os utilizadores a reconhecer, diagnosticar e a recuperar de erros:** Mensagens de erro, devem ser precisas, sugerir uma solução e estar em liguagem natural.

10. **Ajuda e documentação:** Mesmo que a aplicação possa ser usada sem ajuda de documentação ela deve existir e ser facilmente acedida caso os utilizadores tenham duvidas.

Existem quatro formas [21] de testar a usabilidade de um interface:

- **Automaticamente**, ou seja, medido computacionalmente, correndo a especificação do interface de utilizador num programa.

- **Empiricamente**, ou seja, a usabilidade é avaliada através de testes com utilizadores reais, que iram testar o interface interagindo com este.

- **Formalmente**, ou seja, usando modelos e formulas para calcular a usabilidade do interface.

- **Informalmente**, ou seja, baseado nas experiência dos avaliadores.

Os testes automáticos não se aplicam ao estado da arte atual e os formais são difíceis de aplicar e não escalam bem para interfaces grandes. Como tal optou-se pelos testes empíricos com um conjunto de quinze utilizadores.

Nestes testes foi pedido a cada utilizador que interagisse com a aplicação, através de tarefas dadas pelo moderador. Este por sua vez recolhia as informações que eram transmitidas pelo utilizador em cada tarefa executada.

Para a realização destes testes elaborou-se um questionário, que foi divido em três secções:

- **Dados pessoais:** Aqui era pedido o nome e a idade do utilizador, sendo verificado igualmente se o utilizador costuma utilizar *sites* desportivos, de forma a identificar a experiência com *sites* deste género.

- **Dados:** Nesta secção foi registado em que *browser* for efetuado o teste.

- **Tarefas:** Nesta secção encontram-se as tarefas a realizar pelos utilizadores, sendo registado para cada tarefa, se o utilizador a concluiu ou não a tarefa em questão, se a tarefa pode ser melhorada e se achou a tarefa difícil ou não, numa escala de 1 (muito fácil) a 5 (muito difícil)

O questionário de usabilidade encontra-se no apêndice A.

5.1.1 Avaliação tarefas

Durante os testes uma tarefa é considerada concluída com sucesso por parte do utilizador se ele conseguiu realiza-la até ao fim e se usou o caminho esperado. Para o caso de sucesso de uma tarefa foi usada a palavra “Sim”e para o caso de insucesso a palavra “Não”. Estas palavras foram usadas igualmente para representar o caso em que o utilizador sugeriu melhoramentos para a tarefa em questão, sendo em caso afirmativo registada igualmente a melhoria indicada. Para medir o grau de dificuldade de uma tarefa foi definido uma escala de 1 (muito fácil) e 5 (muito difícil).

As tarefas realizadas pelos utilizadores estão representadas na tabela 5.1.1 e a tabela 5.1.1 representa o que era esperado que o utilizador fizesse de modo a concluir a tarefa com sucesso.

Nº tarefa	Descrição tarefa
1	Na página inicial indique sobre qual das equipas foram escritas mais notícias em Julho
2	Na página inicial visualize as notícias escritas sobre o Sporting entre 2010 e a atualidade
3	Na página principal, ver mais detalhes sobre a notícia "Leão quer saber fundamentos do castigo a João Mário"
4	Na notícia "Leão quer saber fundamentos do castigo a João Mário" indicar a data da publicação da notícia, jornal em que foi publicada, reação que as pessoas tiveram à notícia, número de gostos e número de comentários.
5	Na notícia "Leão quer saber fundamentos do castigo a João Mário" visualizar os comentários à mesma.
6	Na página principal procurar uma notícia do twitter e ver detalhes.
7	A partir da página principal ir à página que permite votar em notícias.
8	Na página "Votar" votar numa notícia listada
9	Na página "Votar" votar numa notícia sobre o Sporting
10	Ver as notícias que foram concretizadas.
11	Ver detalhes da notícia "OFICIAL: *Herrera* no *Manchester United*" e dizer quantas pessoas classificaram a notícia como concretizada, dar a sua opinião e voltar à página inicial.
12	Visualizar as notícia não concretizadas.
13	Ver detalhes da "Sporting interessado em *Kachunga*" e dizer quantas pessoas disseram que a notícia não se concretizou, dar a sua opinião.
14	Ver notícias sobre o jogador "André *Carrillo*".
15	Qual a equipa mais referenciada em notícias?
16	Qual o jogador mais referenciado em notícias?
17	Qual o jornal com mais notícias acertadas?
18	Encontrar a primeira notícia do facebook com reação positiva e dizer o respetivo título.

Tabela 5.1: Tarefas do questionário de usabilidade

Nº tarefa	Descrição tarefa
1	O utilizador analisar o gráfico e dizer Benfica.
2	Dado que o gráfico já começa com o período pedido por defeito, bastava ao utilizador clicar na linha do gráfico correspondente ao Sporting, aparecendo então as notícias correspondentes.
3	O utilizador percorrer as notícias usando as setas e ao encontrar a notícia carregar em ver mais.
4	Data: 2015-09-09, Jornal: *Record*, Reação: neutra, Gostos: 25, comentários: 2.
5	O utilizador carregar no "+" que aparece imediatamente a seguir aos comentários
6	O utilizador percorrer as notícias usando as setas e ao encontrar a notícia carregar em ver mais.
7	O utilizador carregar em "Notícias->"Votar".
8	O utilizador clicar em "ver mais" numa das notícias e de seguida carregar na opção que achar correta.
9	O utilizador clicar no símbolo do Sporting e carregar "ver mais" numa das notícias e de seguida carregar na opção que achar correta.
10	O utilizador clicar em "Notícias" ->"concretizadas".
11	O utilizador clicar em "ler mais" e dizer que 10 pessoas disseram que a notícia se concretizou, clicar na opção que achar mais correta e de seguida em "*home*" ou "*Rankores*".
12	O utilizador clicar em "Notícias" ->"Não concretizadas".
13	O utilizador clicar em "ler mais" e dizer que 10 pessoas disseram que a notícia se concretizou, clicar na opção mais correta.
14	O utilizador clicar em "Notícias" ->"de jogadores", de seguida clicar no símbolo do Sporting e depois selecionar a foto do André *Carrillo*.
15	O utilizador clicar em "Estatísticas" e dizer Benfica.
16	O utilizador clicar em "Estatísticas" de seguida no botão "Jogadores" e dizer "Andrés Eduardo Fernández Moreno".
17	O utilizador clicar em "Estatísticas" de seguida no botão "Jornais" e dizer "Mais Futebol".
18	O utilizador clicar em "Redes Sociais" percorrer a lista de notícias usando as setas e dizer o titulo.

Tabela 5.2: Tabela com o que é esperado para cada tarefa

5.1.2 Análise de resultados

Para a realização destes teste foram inquiridos quinze utilizadores com idades compreendidas entre os vinte e um e vinte e nove anos que realizaram as tarefas presentes no questionário apresentado por um moderador. Destes quinze utilizadores entrevistados oito usavam sites com notícias desportivas com regularidade. Todos eles usaram o *browser Google Chrome* para a realização dos testes.

Analisando os resultados obtidos dos questionários dos testes de usabilidade podemos ver que em média, 92% das tarefas foram concretizadas com sucesso e apenas 8% não foram concluidas pelos utilizadores. A figura 5.1 mostra a percentagem de conclusão para cada tarefa realizadas pelos quinze utilizadores.

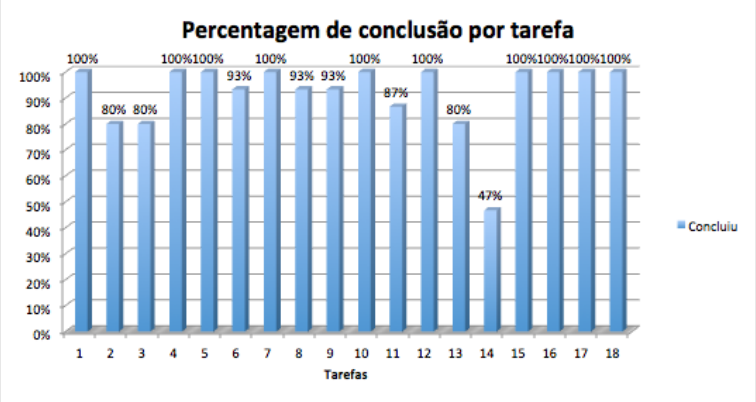

Figura 5.1: Percentagem de conclusão por tarefa

Examinado o gráfico da figura 5.1, verificamos que a tarefa que os utilizadores sentiram mais dificuldade foi a tarefa 14, uma vez que foi concretizada por menos de metade dos utilizadores. Como referido na tabela 5.1.1 nesta tarefa era pedido ao utilizador "Ver notícias sobre o jogador "André *Carrillo*"", analisado os comentários dos utilizadores foi unânime que a principal dificuldade foi que a realização da tarefa dependia do conhecimento do clube a que pertencia o jogador, uma vez que era necessário selecionar primeiro o clube para aparecer as fotos do jogadores desse clube, figura 5.2, e posteriormente escolher a foto do jogador, figura 5.3, cujas notícias se pretendia visualizar. Muitos dos utilizadores que concretizaram a tarefa referiram igualmente que apenas a concretizaram, pois sabiam a que clube pertencia o jogador.

Figura 5.2: Visualizar noticias sobre um jogador - Passo 1

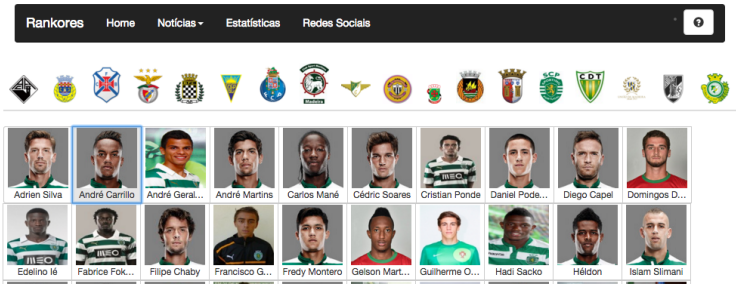

Figura 5.3: Visualizar noticias sobre um jogador - Passo 2

A tarefa 2, não foi concretizada por 20% dos utilizadores, sendo que todos referiram não ter percebido que podiam clicar nas linhas do gráfico figura 5.4.

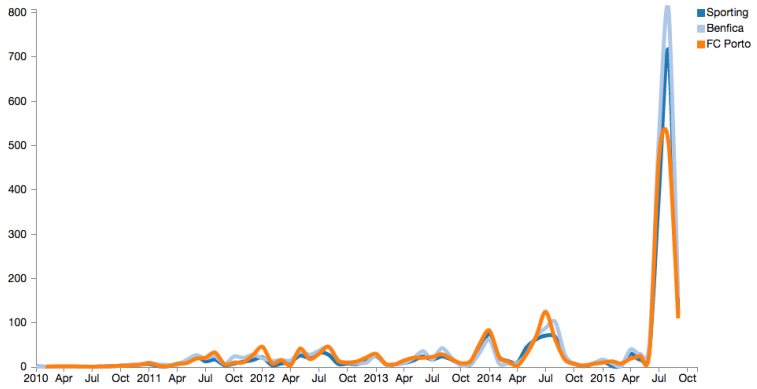

Figura 5.4: Visualizar notícias de uma equipa num período de tempo

A tarefa 3, não foi concretizada por 20% dos utilizadores, sendo que todos eles foram procurar a notícia nas notícias que aparecem após clicar numa das linhas do gráfico figura 5.5, quando era suposto procurarem a mesma no *slideshow* no topo da página figura 5.6.

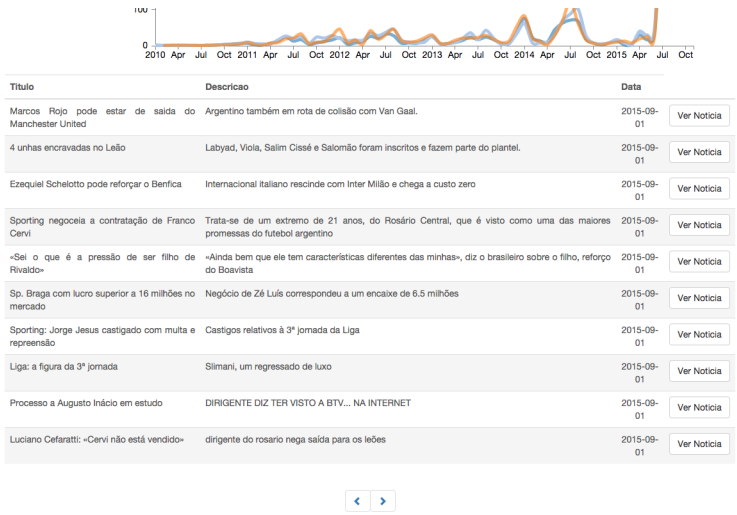

Figura 5.5: Tarefa 3 - encontrar notícia com um dado título - passo errado

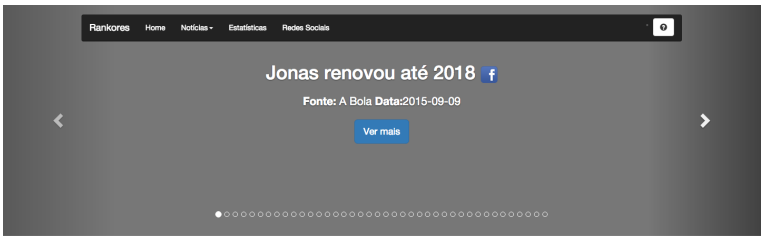

Figura 5.6: Tarefa 3 - encontrar notícia com um dado título - passo esperado

A tarefa 13, não foi concretizada por 20% dos utilizadores, sendo que todos não conseguiram encontrar a notícia, uma vez que era necessário percorrer muitas notícias para encontrar a notícia pedida.

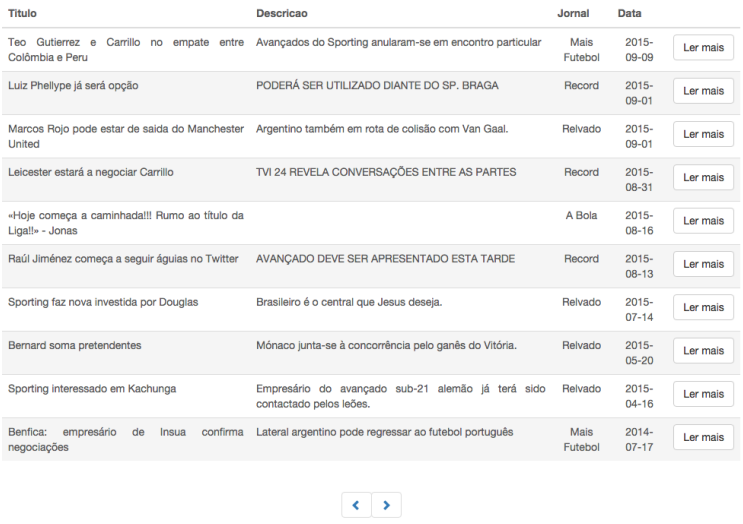

Figura 5.7: Tarefa 13 - encontrar notícia com um dado título

A tarefa 11, não foi concretizada por 13% dos utilizadores,sendo que todos verificaram dificuldades semelhantes às apresentadas na tarefa 13.

Titulo	Descricao	Jornal	Data	
Teo Gutierrez e Carrillo no empate entre Colômbia e Peru	Avançados do Sporting anularam-se em encontro particular	Mais Futebol	2015-09-09	Ler mais
Luiz Phellype já será opção	PODERÁ SER UTILIZADO DIANTE DO SP. BRAGA	Record	2015-09-01	Ler mais
Marcos Rojo pode estar de saída do Manchester United	Argentino também em rota de colisão com Van Gaal.	Relvado	2015-09-01	Ler mais
Leicester estará a negociar Carrillo	TVI 24 REVELA CONVERSAÇÕES ENTRE AS PARTES	Record	2015-08-31	Ler mais
«Hoje começa a caminhada!!! Rumo ao título da Liga!!» - Jonas		A Bola	2015-08-16	Ler mais
Raúl Jiménez começa a seguir águias no Twitter	AVANÇADO DEVE SER APRESENTADO ESTA TARDE	Record	2015-08-13	Ler mais
Sporting faz nova investida por Douglas	Brasileiro é o central que Jesus deseja.	Relvado	2015-07-14	Ler mais
Bernard soma pretendentes	Mónaco junta-se à concorrência pelo ganês do Vitória.	Relvado	2015-05-20	Ler mais
Sporting interessado em Kachunga	Empresário do avançado sub-21 alemão já terá sido contactado pelos leões.	Relvado	2015-04-16	Ler mais
Benfica: empresário de Insua confirma negociações	Lateral argentino pode regressar ao futebol português	Mais Futebol	2014-07-17	Ler mais

Figura 5.8: Tarefa 11 - encontrar notícia com um dado título

Capítulo 6

Conclusão

Analisando os objetivos propostos inicialmente no capitulo 1.2, foi proposto a realização de uma aplicação *Web* que permita validação de notícias sobre transferências futebolísticas por parte da *crowd*.

Com esse objetivo em mente fez-se então o levantamento de requisitos que a aplicação deveria ter e que foram descritos na secção 3.1. Detalhou-se igualmente os casos de uso descrevendo as funcionalidades que a aplicação deveria ter na secção 3.3.

Para implementar essas funcionalidades, criou-se um gráfico usando o *D3.js* para representar a evolução das notícias ao longo do tempo podendo-se mudar o espaço temporal fazendo *scroll* no mesmo.

Foi também criada uma página que permite os utilizadores validarem as notícias como concretizadas ou não concretizadas. Tirando partido das notícias recolhidas implementaram-se filtros de pesquisa com o intuito de permitir ao utilizadores filtrar notícias por equipas e jogadores, através de *clicks* nos símbolos e fotos respetivamente. Implementaram-se também estatísticas que permitem ao utilizador ter uma ideia sobre que equipas e jogadores se escrevem mais notícias. Por fim efetuou-se a integração do sistema com as redes sociais, tirando para isso partido da ferramenta *Semantria* que permite analisar os comentários feitos pelos utilizadores no *Facebook*, classificando-os como positivos, negativos ou neutros. Utilizando essa classificação foi criada igualmente uma página onde são listadas as notícias publicadas nas redes sociais com a respetiva análise de sentimento.

Após a implementação da aplicação, efetuou-se testes de usabilidade com quinze utilizadores. Os resultados destes testes foram positivo, tendo em média, sido concluídas com sucesso 92% das tarefas propostas. No entanto, os utilizadores tiveram dificuldades em algumas tarefas, nomeadamente a tarefa 14, onde apenas 47% dos utilizadores concluiu a mesma.

Olhando para estes resultados, os objetivos propostos foram cumpridos e foi criada um aplicação que permite dizer que notícias foram concretizadas ou não tirando partido do conhecimento da *crowd*.

Relativamente a trabalho a realizar no futuro deverão ser considerados melhorias nos

seguintes pontos:

- o mecanismo de deteção de jogadores e equipas em notícias que neste momento é bastante rudimentar,
- incluir uma barra de pesquisa na aplicação que permita ao utilizadores pesquisar notícias pelo título e pelos jogadores que ocorrem na mesma, respetivamente, algo que neste momento não existe, sendo o filtro das notícias é feito apenas por equipas e jogadores sendo que no caso destes últimos esse filtro é complicado para utilizadores com pouco conhecimento futebolísticos,
- melhorar o *scroll* no gráfico onde são mostradas as notícias ao longo do tempo, uma vez que quando é feito *scroll* para além de este ter de ser feito em cima de uma das linhas do gráfico, este fica desformatado passando as linhas para debaixo do gráfico.

Em suma, este projeto foi bastante interessante e desafiante e que tem um potencial de aplicação a outro tipo de notícias.

Apêndice A

Testes Usabilidade

APLICAÇÕES NA WEB PARA VALIDAÇÃO E CLASSIFICAÇÃO DE NOTÍCIAS PELA CROWD – Testes de Usabilidade

Detalhes Pessoais

Nome:_____ Idade:_____

Dados para o teste

Costuma utilizar sites sobre notícias desportivas?___

Browser Utilizado:

Chrome	**Safari**	**IE**	**Firefox**	**Opera**

Tarefas

De seguida apresenta-se uma lista de tarefas que se pretende que realize, e classifique a dificuldade das mesmas de 1 (muito facil) a 5 (muito dificil). Caso tenha dificuldades na realização consulte o botão de ajuda presente na aplicação e indique qual foi a dificuldade encontrada no local adequado.

Aceder a http://www.lasige.di.fc.ul.pt/webtools/Rankores/ e efetuar as seguintes tarefas.

1. Na página inicial indique sobre qual das equipas foram escritas mais notícias em Julho?
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?___
Se sim como? _____
Esperado: O utilizador analisar o gráfico e dizer Benfica.

2. Na página inicial visualize as notícias escritas sobre o Sporting entre 2010 e a atualidade.
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?___
Se sim como? _____
Esperado: Dado que o gráfico já começa com o período pedido por defeito, bastava ao utilizador clicar na linha do gráfico correspondente ao Sporting, aparecendo então as notícias correspondentes.

3. Na página principal, ver mais detalhes sobre a notícia "Leão quer saber fundamentos do castigo a João Mário".
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___

A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador percorrer as notícias usando as setas e ao encontrar a notícia carregar em ver mais.

4. Na notícia "Leão quer saber fundamentos do castigo a João Mário" indicar a data da publicação da notícia, jornal em que foi publicada, reação que as pessoas tiveram à notícia, número de gostos e número de comentários.
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: Data: 2015-09-09, Jornal: Record, Reação: neutra, Gostos: 25, comentários: 2.

5. Na notícia "Leão quer saber fundamentos do castigo a João Mário" visualizar os comentários à mesma.
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador carregar no "+" que aparece imediatamente a seguir aos comentários.

6. Na página principal procurar uma notícia do *twitter* e ver detalhes.
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador percorrer as notícias usando as setas e ao encontrar a notícia carregar em ver mais.

7. A partir da página principal ir à página que permite votar em notícias.
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador carregar em notícias -> Votar.

8. Na página "Votar" votar numa notícia listada
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____

Esperado: O utilizador clicar em "ver mais" numa das notícias e de seguida carregar na opção que achar correta.
9. Na página "Votar" votar numa notícia sobre o Sporting
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador clicar no símbolo do Sporting e carregar "ver mais" numa das notícias e de seguida carregar na opção que achar correta.

10. Ver as notícias que foram concretizadas.
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador clicar em "Notícias" -> "concretizadas".

11. Ver detalhes da notícia "OFICIAL: Herrera no Manchester United" e dizer quantas pessoas classificaram a notícia como concretizada, dar a sua opinião e voltar à página inicial.
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador clicar em "ler mais" e dizer que 10 pessoas disseram que a notícia se concretizou, clicar na opção que achar mais correta e de seguida em "*home*" ou "*Rankores*".

12. Visualizar as notícia não concretizadas.
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador clicar em "Notícias" -> "Não concretizadas".

13. Ver detalhes da "Sporting interessado em *Kachunga*" e dizer quantas pessoas disseram que a notícia não se concretizou, dar a sua opinião.
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador clicar em "ler mais" e dizer que 10 pessoas disseram que a notícia se concretizou, clicar na opção mais correta.

14. Ver notícias sobre o jogador "André *Carrillo*".
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador clicar em "Notícias" -> "de jogadores", de seguida clicar no símbolo do Sporting e depois selecionar a foto do André *Carrillo*.

15. Qual a equipa mais referenciada em notícias?
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador clicar em "Estatísticas" e dizer Benfica.

16. Qual o jogador mais referenciado em notícias?
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador clicar em "Estatísticas" de seguida no botão "Jogadores" e dizer "Andrés Eduardo Fernández Moreno".

17. Qual o jornal com mais notícias acertadas?
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador clicar em "Estatísticas" de seguida no botão "Jornais" e dizer "Mais Futebol".

18. Encontrar a primeira notícia do *facebook* com reação positiva e dizer o respetivo título.
Concluiu a tarefa?___
Se não, qual foi a dificuldade?_____
A tarefa foi difícil?___
A tarefa pode ser melhorada?____
Se sim como? _____
Esperado: O utilizador clicar em "Redes Sociais" percorrer a lista de notícias usando as setas e dizer o titulo.

Bibliografia

[1] Backbone, the primer. https://github.com/jashkenas/backbone/wiki/Backbone%2C-The-Primer.

[2] Requisitos software. http://www.devmedia.com.br/introducao-a-requisitos-de-software/29580.

[3] AngularJS. https://docs.angularjs.org/guide.

[4] Backbone.js. http://backbonejs.org/.

[5] Matko Boanjak, Eduardo Oliveira, José Martins, Eduarda Mendes Rodrigues, and Luís Sarmento. Twitterecho: A distributed focused crawler to support open research with twitter data. In *Proceedings of the 21st International Conference Companion on World Wide Web*, WWW '12 Companion, pages 1233–1240, New York, NY, USA, 2012. ACM.

[6] Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1121–1128, 2009.

[7] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, 2011.

[8] Nicholas Chase. Understanding web services specifications, part 1: Soap. http://www.ibm.com/developerworks/webservices/tutorials/ws-understand-web-services1/ws-understand-web-services1.html.

[9] Junghoo Cho and Hector Garcia-Molina. The evolution of the web and implications for an incremental crawler. 1999.

[10] Graham Cormode and Balachander Krishnamurthy. Key differences between web 1.0 and web 2.0. *First Monday*, 13(6), 2008.

[11] Itay Herskovits. Angularjs vs backbonejs. http://blog.backand.com/angular-vs-backbone/.

[12] Infragistics. Angularjs vs. backbone.js vs ember.js. http://www.infragistics.com/community/blogs/marketing/archive/2015/01/27/angularjs-vs-backbone-js-vs-ember-js.aspx.

[13] Nicholas Kushmerick. *Wrapper induction for information extraction*. PhD thesis, University of Washington, 1997.

[14] Avraham Leff and James T Rayfield. Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, pages 118–127. IEEE, 2001.

[15] Fernando Mantoan. Série backbone.js: Parte 1 - introdução. http://blog.fernandomantoan.com/serie-backbone-js-parte-1-introducao/.

[16] Ali Mesbah and Arie Van Deursen. Migrating multi-page web applications to single-page ajax interfaces. In *Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on*, pages 181–190. IEEE, 2007.

[17] Jussi Myllymaki. Effective web data extraction with standard xml technologies. *Computer Networks*, 39(5):635–644, 2002.

[18] Sabine Niederer and José Van Dijck. Wisdom of the crowd or technicity of content? wikipedia as a sociotechnical system. *New Media & Society*, 12(8):1368–1387, 2010.

[19] Jakob Nielsen. 10 usability heuristics for user interface design. http://www.nngroup.com/articles/ten-usability-heuristics/.

[20] Jakob Nielsen. Usability 101: Introduction to usability. http://www.nngroup.com/articles/usability-101-introduction-to-usability/.

[21] Jakob Nielsen. Usability inspection methods. In *Conference Companion on Human Factors in Computing Systems*, CHI '94, pages 413–414, New York, NY, USA, 1994. ACM.

[22] Oracle. What are restful web services? https://docs.oracle.com/javaee/6/tutorial/doc/gijvh.html.

[23] Semantria. https://semantria.com/support/developer.

[24] Bhavin Shah. Scalable vector graphics. In *Conference on Multimedia Systems*, 2003.

[25] Uri Shaked. Angularjs vs. backbone.js vs. ember.js. `https://www.airpair.com/js/javascript-framework-comparison`.

[26] Underscore.js. `http://underscorejs.org/`.

[27] Aaron Wheeler. Is underscore.js + backbone.js + jquery = prototype.js? `https://www.quora.com/Is-Underscore-js-+-Backbone-js-+-jQuery-Prototype-js`.

[28] Sheng Kung Michael Yi, Mark Steyvers, Michael D Lee, and Matthew J Dry. The wisdom of the crowd in combinatorial problems. *Cognitive science*, 36(3):452–470, 2012.